Michael Waidner

Byzantinische Verteilung ohne kryptographische Annahmen trotz beliebig vieler Fehler

© Michael Waidner, 1992
Universität Karlsruhe, Institut für Rechnerentwurf und Fehlertoleranz
Postfach 6980, W7500 Karlsruhe 1, Germany
Tel.: ++49-721-608-4024, Fax.: ++49-721-370455, email: waidner@ira.uka.de

Byzantinische Verteilung ohne kryptographische Annahmen trotz beliebig vieler Fehler

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

der Fakultät für Informatik
der Universität Karlsruhe (Technische Hochschule)

genehmigte

Dissertation

von

Michael Waidner
aus Mühlacker

Tag der mündlichen Prüfung:	24. 10. 1991
Erster Gutachter:	Prof. Dr.-Ing. W. Görke
Zweiter Gutachter:	Prof. Dr. R. Reischuk

Abstract

For PEASE/SHOSTAK/LAMPORT's *Byzantine Generals Problem* (also called *Reliable Broadcast Problem*), it is investigated how many faulty processors can be tolerated in the original Byzantine failure model. It is assumed that all correct processors are synchronized and are able to communicate reliably with each other. Up to t of all n processors may become maliciously faulty. There are no computational assumptions (like "signatures are unforgeable") on the behaviour of faulty processors.

On these assumptions, many protocols are known which tolerate any number $t < n/3$ of faulty processors, and it is well known that for $t \geq n/3$, there are no deterministic protocols.

In the following, for the first time, protocols are described which tolerate $n/3$ and more faulty processors in the Byzantine failure model: They have an error probability $\leq 2^{-\sigma}$ and tolerate any $t < n/2$ in a constant expected number of rounds (worst case in $O(\sigma)$ rounds), or any $t < n$ in $t+1$ rounds. Their computational and communication complexity is polynomial in n and σ.

For all our protocols, it is necessary that an arbitrary time before they are executed a preparation protocol is performed during which each processor broadcasts some information to all other processors reliably (similar to the key distribution of digital signatures).

Basis of our protocol tolerating any $t < n/2$ is a verifiable secret sharing scheme which needs reliable broadcast for sharing a secret, but not for revealing it. Basis of our protocol tolerating any $t < n$ is a new unconditionally secure authentication technique, which can replace computationally secure digital signatures in many authenticated protocols.

Kurzfassung

Für das von PEASE, SHOSTAK und LAMPORT 1980 formulierte Problem der byzantinischen Verteilung (auch Problem der byzantinischen Generäle oder zuverlässige Verteilung genannt) wird untersucht, wieviele fehlerhafte Prozessoren im ursprünglichen byzantinischen Fehlermodell toleriert werden können. Es wird dabei angenommen, alle korrekten Prozessoren seien synchronisiert und können zuverlässig paarweise miteinander kommunizieren. Bis zu t der n Prozessoren können bösartig fehlerhaft werden. Über das Verhalten der fehlerhaften Prozessoren werden keine Annahmen gemacht (auch nicht die, daß sie keine Signaturen fälschen können).

Unter diesen Voraussetzungen sind viele Protokolle bekannt, die jede Anzahl $t < n/3$ tolerieren können, und es ist bekannt, daß für $t \geq n/3$ keine deterministischen Protokolle existieren.

In dieser Arbeit werden erstmals Protokolle beschrieben, die im byzantinischen Fehlermodell $n/3$ und mehr fehlerhafte Prozessoren tolerieren: Sie haben jeweils eine Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ und tolerieren jedes $t < n/2$ in einer erwartet konstanten Anzahl an Runden (im schlimmsten Fall in $O(\sigma)$ Runden) oder jedes $t < n$ in $t+1$ Runden. Ihr Rechen- und Nachrichtenaufwand ist polynomial in n und σ.

Alle diese Protokolle setzen voraus, daß beliebige Zeit vor ihrer Ausführung ein Vorbereitungsprotokoll ausgeführt wird, in dem jeder Prozessor zuverlässig Nachrichten an alle anderen Prozessoren verteilen kann (ähnlich der Schlüsselverteilung von digitalen Signaturen).

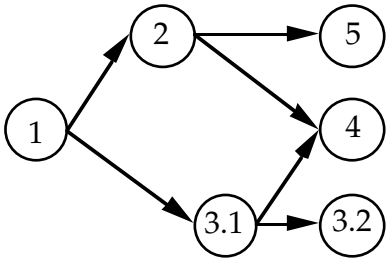
Die Grundlage des Protokolls für $t < n/2$ ist ein Verfahren zur überprüften Geheimnisaufteilung (*verifiable secret sharing*), das zwar zum Aufteilen, nicht aber zum Zusammensetzen eines Geheimnisses einen zuverlässigen Verteilkanal benötigt. Die Grundlage des Protokolls für $t < n$ ist ein neuartiger, unbedingt sicherer Authentikationsmechanismus, der in vielen authentisierten Protokollen die nur kryptographisch sicheren digitalen Signaturen ersetzen kann.

Hinweise für den Leser

Alle für diese Arbeit wesentlichen Kenntnisse über byzantinische Verteilung sind in Kapitel 1 zusammengefaßt. Wer sich weitergehend mit dem Thema befassen möchte, sei auf die Überblicksarbeiten [ChDw_89, Fisc_83, Reis_87] verwiesen.

Überblicke über die Ergebnisse der Arbeit werden in Abschnitt 1.5 und in Kapitel 6 gegeben.

Die Abhängigkeiten der Kapitel sind in der folgenden Graphik verdeutlicht. Ein Pfeil von x nach y bedeutet, daß der Inhalt von Kapitel x Voraussetzung für das Verständnis von Kapitel y ist.



Sätze und Lemmata sind kapitelweise gemeinsam fortlaufend durchnumeriert, d. h. Satz/Lemma $x.y$ ist der y-te Satz oder Lemma in Kapitel x. Literaturverweise im Text sind stets alphabetisch angeordnet.

Inhalt

Das historische Problem der byzantinischen Verteilung 1

1 Byzantinische Verteilung 3

1.1 Einleitung 4
1.1.1 Problemstellung 4
1.1.2 Bekannte Lösungen 6
1.1.2.1 Lösungen im byzantinischen Fehlfunktionsmodell 7
1.1.2.2 Authentisierte Verteilungsprotokolle 7
1.1.2.2.1 Authentikation im byzantinischen Fehlfunktionsmodell 8
1.1.2.2.2 Digitale Signaturen im kryptographischen Fehlfunktionsmodell 9
1.1.2.3 Kombination von nichtauthentisierten und authentisierten Verteilungsprotokollen 11
1.1.2.4 Schwächere Fehlfunktionsmodelle 12
1.2 Verteiltes System mit Fehlern 12
1.2.1 Das korrekte System 13
1.2.2 Das byzantinische Fehlermodell 15
1.2.3 Qualität eines Protokolls 17
1.3 Bekannte Grenzen der byzantinischen Verteilung 19
1.4 Diskussion der Annahmen bezüglich Synchronisation und Kommunikation 22
1.4.1 Synchronisation 22
1.4.2 Sichere und nichtabhörbare Verbindungen 25
1.4.3 Vorbereitungsprotokoll mit zuverlässigem Verteilkanal 26
1.4.4 Fehlerwahrscheinlichkeit 27
1.5 Überblick über die Arbeit 28

2 Byzantinische Verteilung mit korrekter Mehrheit in erwartet konstanter Rundenanzahl 30

2.1 Überprüfte Geheimnisaufteilung 30
2.1.1 Überblick über überprüfte Geheimnisaufteilung 31

2.1.2 RABINs Protokoll zur überprüften Geheimnisaufteilung 33
2.1.2.1 SHAMIRs Geheimnisaufteilungsprotokoll 34
2.1.2.2 RABINs Authentikationsnachweisprotokoll 35
2.1.2.3 RABINs Geheimnisfestlegungsprotokoll 40
2.1.2.4 RABINs Protokoll zur überprüften Geheimnisaufteilung 43
2.1.2.5 Einsparung von Iterationen der Verifikationsprozedur im Protokoll zur überprüften Geheimnisaufteilung 52
2.1.3 Überprüfte Geheimnisaufteilung ohne zuverlässigen Verteilkanal beim Zusammensetzen 53
2.1.3.1 Schwache Geheimnisfestlegung 53
2.1.3.2 Überprüfte Geheimnisaufteilung 55
2.1.3.3 Wahl der Protokollparameter 57
2.2 Byzantinische Verteilung mit korrekter Mehrheit 59
2.2.1 Byzantinische Verteilung mit Fehlerwahrscheinlichkeit $\approx 1/L$ in $O(L)$ Runden 59
2.2.2 Byzantinische Verteilung mit Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ in $O(\sigma)$ Runden 66
2.2.3 Byzantinische Verteilung mit Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ in erwartet 19 Runden 69
2.3 Zusammenfassung 77

3 Byzantinische Verteilung mit beliebig vielen Störern in linearer Rundenanzahl 78

3.1 Pseudosignaturen 78
3.1.1 Grundidee 79
3.1.2 Minisignaturen durch Authentikationscodes 81
3.1.3 Das Initialisierungsprotokoll für Pseudosignaturen 82
3.1.3.1 Das DC-Protokoll 82
3.1.3.2 Mehrfachzugriff für das DC-Protokoll nach BOS / den BOER 84
3.1.3.3 Initialisierungsprotokoll 88
3.1.3.3.1 Erkennen von Störungen im DC-Protokoll 89
3.1.3.3.2 Fehlerlokalisierung und Rekonfigurierung 92
3.1.4 Pseudosignaturen mit λ Akzeptionsstufen 98
3.1.5 Sicherheit gegen passive Angriffe 100

3.1.6 Sicherheit gegen aktive Angriffe ... 103
3.1.6.1 Ordentliche Protokolle ... 104
3.1.6.2 Korrektheit ... 105
3.1.6.3 Aktive Angriffe auf die Unfälschbarkeit ... 105
3.1.6.4 Aktive Angriffe auf die Weitergebbarkeit ... 106
3.2 Byzantinische Verteilung mit Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ in t+1 Runden ... 109
3.2.1 Modifiziertes DOLEV/STRONG-Protokoll ... 109
3.2.2 Beweis des byzantinischen Verteilungsprotokolls ... 112
3.2.3 Polynomial viele Verteilungen mit logarithmischer Vorbereitung ... 117
3.3 Zusammenfassung ... 123

4 Erweiterung binärer Verteilung auf lange Nachrichten ... 124
4.1 Byzantinische Pseudoverteilung ... 124
4.2 TOUEGs Protokoll ... 127
4.3 Polynomial viele Verteilungen mit logarithmischer Vorbereitung ... 132
4.4 Zusammenfassung ... 135

5 Anwendung auf Mehrparteienberechnungen ... 136
5.1 Definitionen und Fakten ... 136
5.2 Anwendung der byzantinischen Verteilungsprotokolle ... 138

6 Zusammenfassung ... 140

Häufig verwendete Symbole ... 144
Stichwörter ... 146
Literatur ... 149

Das historische Problem der byzantinischen Verteilung

Das Problem der byzantinischen Verteilung ist eines der wenigen wirklich klassischen Probleme der Informatik: [LaSP_82] beschreibt die entsprechenden Nöte byzantinischer Generäle bei der Belagerung einer feindlichen Stadt. In noch viel größerem Ausmaß hatten dieses Problem sogar schon mindestens 380 Jahre zuvor römische Offiziere!

Bild 0-1 Belagerung eines gallischen Dorfes durch römische Zenturien

Im Jahre 50 v. Chr. wurde ganz Gallien von den Römern beherrscht — mit Ausnahme eines kleinen Dorfes (vgl. Bild 0-1). Dort hatte der geniale Pharmazeut MIRACULIX einen Zaubertrank entwickelt, der ungeheuere Kräfte verlieh. Mit diesem gut versorgt, konnten die gallischen Dorfbewohner leicht gegen die ansonsten unbezwingbaren römischen Eroberer standhalten [GoUd_61].

Natürlich störte den damaligen Regierungschef G. JULIUS CAESAR dieses letzte freie gallische Dorf ganz besonders. Er entsandte daher einige Zenturionen mit

ihren Legionären unter dem Oberbefehl eines erfahrenen Generals, um dem Unfug ein Ende zu setzen.

Die Zenturien verschanzten sich rund um das Dorf, jede in einem eigenen, gut befestigten Lager, und warteten auf eine günstige Gelegenheit zum Angriff. Nach langer Zeit unerquicklicher Belagerung erwarteten die Zenturionen endlich den Befehl des Generals, „angreifen!“ oder „abziehen!“. Seine Entscheidung konnte der General aber nicht, wie sonst üblich, in einer Versammlung aller Zenturionen bekanntgeben. Ohne ihre Anführer hätten die Legionäre die Belagerung sicher vergessen und wären nach Hause gegangen. Also mußte der General seinen Befehl durch Boten überbringen lassen.

Die Zenturionen trauten dem überbrachten Befehl aber nicht. Der General war ein berüchtigter Lebemann, und es war durchaus möglich, daß er sich durch einige edle Flaschen Zaubertranks hätte bestechen lassen, widersprüchliche Befehle auszuteilen. Hätten aber manche Zenturien angegriffen, während andere den Rückzug angetreten hätten, so hätten die Gallier ein noch leichteres Spiel mit ihnen gehabt.

Die Zenturionen hatten sich daher vorsorglich vorgenommen, die erhaltenen Befehle des Generals zuerst untereinander auszutauschen. War der General doch ehrlich, so sollte sein Befehl auf diese Art bestätigt werden. Ansonsten wollten sie sich wenigstens auf einen einheitlichen Befehl einigen, gleichgültig welchen. Die Zenturionen trauten aber auch einander nicht. Selbst die Echtheit gesiegelter und unterschriebener Briefe erschien ihnen fraglich, denn sie wußten, ihre Gegner konnten mit Hilfe von MIRACULIX jedes noch so kunstvolle Siegel und jede noch so schwierige Unterschrift täuschend echt nachahmen.

Es ist leider nicht überliefert, wie die römischen Belagerer damals ihr Problem lösten. Die informatische Forschung hat seither aber glücklicherweise einige Fortschritte gemacht.

Man hat Lösungen gefunden, die erfolgreich gearbeitet hätten, wenn mehr als zwei Drittel aller Offiziere ehrlich gewesen wären, und alternativ auch solche, die erfolgreich gearbeitet hätten, wenn trotz MIRACULIX Hilfe keine Siegel und Unterschriften hätten gefälscht werden können [LaSP_82]. Lösungen, die unter schwächeren Bedingungen erfolgreich gearbeitet hätten, waren bislang unbekannt.

Nichtsdestotrotz werden im folgenden solche Lösungen vorgestellt: Sie arbeiten auch angesichts eines unbeschränkt mächtigen Gegners wie MIRACULIX und tolerieren, daß fast die Hälfte (Kapitel 2, Kapitel 4) oder sogar fast alle (Kapitel 3) Offiziere unehrlich sind. Erkauft wird die erhöhte Sicherheit dadurch, daß die Offiziere vor Beginn der Belagerung, etwa vor ihrer Entsendung in Rom, sich treffen und einige Informationen austauschen müssen, und dadurch, daß mit einer beliebig (exponentiell) kleinen Wahrscheinlichkeit doch keine Übereinstimmung erzielt wird.

1 Byzantinische Verteilung

Das Problem der *byzantinischen Verteilung* ist eines der grundlegenden Probleme aus dem Bereich der *verteilten, fehlertoleranten Algorithmen*.

Ein *verteilter* Algorithmus ist einer, der auf mehreren miteinander via Nachrichten kommunizierenden Rechnern (im folgenden *Prozessoren* genannt) ausgeführt wird. Ein verteilter Algorithmus ist *fehlertolerant*, wenn er trotz einer beschränkten Anzahl fehlerhafter Komponenten (Prozessoren, Verbindungen) seine spezifizierte Aufgabe löst.

Das Problem der *byzantinischen Verteilung* besteht, grob gesagt, darin, auf einem Kommunikationsnetz, das nur paarweise Kommunikation erlaubt, ein *zuverlässiges* Verteilnetz zu simulieren. Ein solches garantiert auch für den Fall eines fehlerhaften *Transmitters*, daß stets alle Prozessoren die gleichen Nachrichten erhalten.

Byzantinische Verteilungsprotokolle sind Lösungen dieses Problems und werden überall da benötigt, wo in einem fehlertoleranten verteilten System alle Beteiligten eine korrekte globale Systemsicht haben sollen.

Eine der wesentlichen Fragestellungen ist, wie *fehlertolerant* ein byzantinisches Verteilungsprotokoll im besten Fall sein kann.

Zur Beantwortung dieser Frage wird hier angenommen, alle korrekten Prozessoren seien synchronisiert und können jeweils paarweise zuverlässig miteinander kommunizieren, aber bis zu t aller n Prozessoren können fehlerhaft werden. Über das fehlerhafte Verhalten von Prozessoren werden keine Annahmen gemacht (*byzantinisches Fehlfunktionsmodell*).

In diesem sogenannten *byzantinischen Fehlermodell* waren bislang nur Lösungen bekannt, die $t < n/3$ fehlerhafte Prozessoren tolerieren, und es war auch bekannt, daß für $t \geq n/3$ keine deterministischen Lösungen existieren.

In dieser Arbeit werden erstmals byzantinische Verteilungsprotokolle vorgestellt, die mit einer Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ jedes $t < n/2$ in erwartet $O(1)$ und maximal $O(\sigma)$ Runden bzw. jedes $t < n$ in $O(t)$ Runden tolerieren. Der Aufwand ist jeweils polynomial in σ und n.

Diese Lösungen setzen voraus, daß (beliebige Zeit) vor dem Verteilungsprotokoll ein *Vorbereitungsprotokoll* ausgeführt wird, und daß in diesem jeder Prozessor zuverlässig Nachrichten an alle verteilen kann.

Im Rest dieses Kapitels 1 wird das Problem der byzantinischen Verteilung genauer definiert und beschrieben, was bislang über die Lösbarkeit des Problems bekannt war.

In Abschnitt 1.1 wird byzantinische Verteilung definiert und werden die

wichtigsten bekannten Lösungen aufgezählt. Abschnitt 1.2 beschreibt das im folgenden verwendete byzantinische Fehlermodell. In Abschnitt 1.3 wird zusammengefaßt, welche Grenzen für die Fehlertoleranz eines Verteilungsprotokolls bekannt sind. Die für den Rest der Arbeit gemachten Annahmen des byzantinischen Fehlermodells werden in Abschnitt 1.4 diskutiert. Abschnitt 1.5 gibt einen genaueren Überblick über die Ergebnisse der Arbeit.

1.1 Einleitung

1.1.1 Problemstellung

Betrachtet wird ein verteiltes System aus n **Prozessoren**, die jeweils nur paarweise miteinander kommunizieren können, d. h. nur über ein (evtl. virtuelles) vollvermaschtes Punkt-zu-Punkt-Netz. Die Prozessoren arbeiten synchronisiert, die Übertragung einer Nachricht erfolgt zuverlässig und in bekannt begrenzter Zeit. Erhält ein Prozessor eine Nachricht, so kann er den unmittelbaren Sender zweifelsfrei erkennen.

Von diesen n Prozessoren wird einer ausgezeichnet und als **Transmitter** bezeichnet. Dieser hat einen Anfangswert v aus einer Nachrichtenmenge mit wenigstens zwei Elementen.

Das **byzantinische Verteilungsproblem** besteht nun darin, diesen Anfangswert v zuverlässig an alle Prozessoren zu verteilen [PeSL_80]:

Terminierung	Nach einer a priori bekannten endlichen Zeit akzeptiert jeder Prozessor genau einen *Endwert*.
Korrektheit	War der Transmitter korrekt und hatte den Anfangswert v, so akzeptiert jeder korrekte Prozessor v als Endwert.
Konsistenz	Alle korrekten Prozessoren akzeptieren denselben Endwert.

Auf dem vollvermaschten Punkt-zu-Punkt-Netz soll also eine Nachricht zuverlässig verteilt werden.

(Im historischen Beispiel entsprachen die Offiziere den Prozessoren, der General dem Transmitter und der Befehl des Generals dem zu verteilenden Wert.)

Das byzantinische Verteilungsproblem wurde in [LaSP_82] als das *Byzantine Generals Problem* bezeichnet. Häufig wird es auch *Byzantine Agreement* [Dole_82] oder *Reliable Broadcast* genannt [Fisc_83, Reis_87].

Ein Protokoll zur Lösung des Problems heißt (**byzantinisches**) **Verteilungsprotokoll**. Ein **binäres Verteilungsprotokoll** kann nur binäre Nachrichten verteilen, d. h. Nachrichten aus {0,1}.

Problematisch wird die byzantinische Verteilung natürlich erst dann, wenn Prozessoren, den Transmitter eingeschlossen, **fehlerhaft** werden können (im

historischen Beispiel dann, wenn Offiziere zu Verrätern werden).

Um dies einzusehen, betrachte man das einfachste Verteilungsprotokoll: Der Transmitter sendet seinen Anfangswert v an alle Prozessoren, und jeder Prozessor akzeptiert den erhaltenen Wert sofort als seinen Endwert. Wird hier der Transmitter fehlerhaft und versendet verschiedene Nachrichten, so ist offensichtlich die Konsistenz verletzt. Ein solcher Fehler wird häufig **byzantinischer Fehler** genannt.

Fehlerhafte Prozessoren werden im folgenden **Störer** genannt.

Die **Fehlertoleranz** eines Verteilungsprotokolls wird, wie üblich, durch ein Fehlfunktionsmodell und durch die maximal tolerierbare Anzahl t an Störern beschrieben. Das Fehlfunktionsmodell beschreibt das tolerierbare Verhalten der Störer. Die Kommunikation zwischen Prozessoren wird als fehlerfrei angenommen, d. h. nicht tolerierte Verbindungsfehler werden implizit als Prozessorfehler gedeutet.

In dieser Arbeit wird in diesem Sinne ein Maximum an Fehlertoleranz angestrebt. Daher wird *kein* einschränkendes Fehlfunktionsmodell verwendet: Störer können im schlimmsten Fall einem beliebigen Protokoll folgen (**byzantinisches Fehlfunktionsmodell** [ChDw_89]). Hier sind keine Fehlertoleranzmaßnahmen anwendbar, die darauf aufbauen, Störer verhielten sich gutartig (z. B. stören = anhalten) oder seien zumindest zu sehr komplexen Berechnungen (z. B. Brechen eines Kryptosystems) nicht fähig. Praktisch mag man sich daher im folgenden unter den Störern viele kooperierende und für den Protokollentwerfer nicht einschätzbare Saboteure vorstellen.

Ausführlicher wird das byzantinische Fehlermodell in Abschnitt 1.2 beschrieben.

Um Verwechslungen zu vermeiden, seien noch zwei ebenfalls häufig *Byzantine Agreement* genannte Probleme erwähnt:

Agieren alle Prozessoren zugleich als Transmitter, d. h. hat *jeder* Prozessor einen Anfangswert, der an alle zuverlässig verteilt werden soll, so erhält man das Problem der **interaktiven Übereinstimmung** (*interactive consistency*) [Fisc_83, PeSL_80, Reis_87].

Ähnlich definiert ist das **Übereinstimmungsproblem** (*consensus problem*) [Fisc_83]: Jeder Prozessor hat einen eigenen Anfangswert. Nach endlicher Zeit akzeptiert jeder korrekte Prozessor *einen* Endwert, für den gilt:

Korrektheit	Hatten alle korrekten Prozessoren denselben Anfangswert v, so akzeptiert jeder korrekte Prozessor v als Endwert.
Konsistenz	Alle korrekten Prozessoren akzeptieren denselben Endwert.

Das Problem der byzantinischen Verteilung und das der interaktiven Übereinstimmung sind offensichtlich äquivalent, d. h. mit einem Protokoll zur Lösung

des einen Problems kann man leicht ein Protokoll zur Lösung des anderen Problems konstruieren [Fisc_83].

Das Übereinstimmungsproblem ist höchstens dann lösbar, wenn eine Mehrheit der Prozessoren korrekt ist, wie man sich leicht klar macht:

Angenommen, man habe eine Lösung für $n = 2t$, und es seien auch wirklich genau t Prozessoren fehlerhaft. Als Werte treten nur „0“ und „1“ auf. Das fehlerhafte Verhalten der Störer beschränke sich darauf, daß sie ihren Anfangswert frei wählen, und zwar gerade so, daß insgesamt jeweils t Prozessoren als Anfangswert den Wert „0“ bzw. „1“ haben. Hat ein korrekter Prozessor den Anfangswert b, so kann er nicht ausschließen, daß genau die t Prozessoren, die denselben Anfangswert haben, die korrekten sind. Folglich muß er seinen Anfangswert b als Endwert akzeptieren. Gibt es aber zwei korrekte Prozessoren mit verschiedenen Anfangswerten, so gibt es auch zwei korrekte Prozessoren, die verschiedene Endwerte akzeptieren, im Widerspruch zur Konsistenzforderung.

Für $n = 2t$ kann es auch keine Lösung geben, die mit einer *beliebig* klein wählbaren Fehlerwahrscheinlichkeit arbeitet (im Gegensatz zum byzantinischen Verteilungsproblem): Um die obige Situation herbeiführen zu können, müssen die Störer nur wissen, wieviele der t korrekten Prozessoren den Wert „0“ hatten, und mit Wahrscheinlichkeit $\geq 1/t$ können sie diesen Wert richtig erraten.

Eine korrekte Mehrheit ($n > 2t$) vorausgesetzt, sind byzantinisches Verteilungs- und Übereinstimmungsproblem äquivalent [Fisc_83]:

Gegeben eine Lösung des Übereinstimmungsproblems für $t < n/2$: Der General sendet seinen Wert zuerst direkt an alle Prozessoren, die dann über diesen erhaltenen Wert als Anfangswert Übereinstimmung erzielen.

Gegeben eine Lösung des byzantinischen Verteilungsproblems für $t < n/2$: Dann verteilt jeder Prozessor zuverlässig seinen Anfangswert. Alle erhalten übereinstimmend den Vektor aller n Anfangswerte. Kommt in diesem ein Wert v mehrheitlich vor (d. h. mit Vielfachheit $> n/2$), so akzeptiert jeder v als Endwert; ansonsten akzeptieren alle einen beliebig vordefinierten Standardwert. Die Konsistenz ist natürlich gesichert. Haben alle korrekten Prozessoren denselben Anfangswert v, so taucht dieser Wert wenigstens $n-t$ mal im Vektor auf. Wegen $t < n/2$ ist $n-t > n/2$, d. h. alle akzeptieren diesen Wert, und auch die Korrektheit ist gesichert.

Weitere Varianten des byzantinischen Verteilungsproblems sind in den Übersichtsarbeiten [ChDw_89, Fisc_83, Reis_87] beschrieben.

1.1.2 Bekannte Lösungen

Im folgenden wird, sortiert nach den verwendeten Fehlfunktionsmodellen, ein kurzer Überblick über bekannte Lösungen des byzantinischen Verteilungs-

problems gegeben. Alle aufgeführten Lösungen gehen von den in Abschnitt 1.1.1 genannten Voraussetzungen hinsichtlich Synchronisation und Kommunikation aus.

1.1.2.1 Lösungen im byzantinischen Fehlfunktionsmodell

Das byzantinische Fehlfunktionsmodell ist das stärkste mögliche, d. h. es gibt kein Fehlfunktionsmodell, das eine Fehlfunktion beschreibt, die nicht auch im byzantinischen Fehlfunktionsmodell enthalten wäre.

Für das byzantinische Fehlfunktionsmodell waren bislang nur Verteilungsprotokolle bekannt, die ein beliebiges $t < n/3$ tolerieren konnten [DFFL_82, PeSL_80, Überblick: ChDw_89, Fisc_83, Reis_87]. Im deterministischen Sinne ist für $t \geq n/3$ das Problem unlösbar [PeSL_80]. Ausführlicher wird hierauf in Abschnitt 1.3 eingegangen.

1.1.2.2 Authentisierte Verteilungsprotokolle

Authentisierte Verteilungsprotokolle gehen von der Existenz eines **sicheren Authentikationsmechanismus** aus:[1]

Korrektheit Jeder korrekte Prozessor P kann jede Nachricht so *authentisieren*, daß jeder korrekte Prozessor Q sie als von P authentisiert *akzeptiert*.

Unfälschbarkeit Akzeptiert ein korrekter Prozessor Q eine Nachricht als von P authentisiert und ist P korrekt, so hat P sie zuvor authentisiert.

Weitergebbarkeit Akzeptiert ein korrekter Prozessor Q eine Nachricht als von P authentisiert, so kann er sie so an jeden korrekten Prozessor R weitergeben, daß auch dieser sie als von P authentisiert akzeptiert.

Authentisierte Verteilungsprotokolle können im besten Fall beliebig viele Störer tolerieren, d. h. jedes $t < n$ [DoSt_83, PeSL_80].

Um ein authentisiertes Verteilungsprotokoll verwenden zu können, muß allerdings erst ein sicherer Authentikationsmechanismus gefunden werden.

Im byzantinischen Fehlfunktionsmodell gab es bislang für $t \geq n/3$ *keinen* geeigneten Authentikationsmechanismus (Abschnitt 1.1.2.2.1). Die üblicherweise als Authentikationsmechanismus vorgeschlagenen digitalen Signaturen erfüllen

---

[1] Die folgenden drei Forderungen an einen Authentikationsmechanismus sind zwar üblich [PeSL_80], werden aber meist nicht explizit genannt. Zugeschnitten auf einen speziellen Authentikationsmechanismus sind sie in [SrTo_87] enthalten.

zwar die obigen Anforderungen, allerdings höchstens im schwächeren kryptographischen Fehlfunktionsmodell (Abschnitt 1.1.2.2.2).

In Abschnitt 3.1 wird ein Authentikationsmechanismus vorgestellt werden, der für viele authentisierte Protokolle geeignet sicher ist.

1.1.2.2.1 Authentikation im byzantinischen Fehlfunktionsmodell

Alle bislang bekannten Authentikationsmechanismen, die im byzantinischen Fehlfunktionsmodell eingesetzt werden können, sind zur Implementierung authentisierter Protokolle für $t \geq n/3$ ungeeignet.

Speziell für die Implementierung authentisierter Protokolle ist ein Authentikationsmechanismus aus [SrTo_87] gedacht, der dort kurz ***Broadcast Primitive*** genannt wird. Er setzt $t < n/3$ voraus und simuliert die (unzuverlässige) Verteilung einer authentisierten Nachricht. Der Mechanismus garantiert die Korrektheit und Unfälschbarkeit. Die Weitergebbarkeit wird in der folgenden Form garantiert: Akzeptiert ein korrekter Prozessor eine authentisierte Nachricht in Runde r, so wird jeder andere korrekte Prozessor diese Nachricht spätestens in Runde $r+1$ akzeptiert haben. Dies genügt für viele authentisierte Protokolle [SrTo_87, SrTo1_87], z. B. für das authentisierte Verteilungsprotokoll aus [DoSt_83].

Authentikationscodes erfüllen nur die Forderungen nach Korrektheit und Unfälschbarkeit [GiMS_74]. Eine authentisierte Nachricht kann jedoch nicht weitergegeben werden: Erhält ein Prozessor Q eine von P authentisierte Nachricht, so kann er keinen dritten Prozessor R von der Authentizität der Nachricht überzeugen.

Durch **Authentikationsnachweise** (*information checking*) kann eine Nachricht zusätzlich *einmal* weitergegeben werden [RaBe_89]:

Ein Prozessor P kann eine Nachricht N so an einen Prozessor Q senden, daß Q sie später an einen dritten Prozessor R weitergeben und R die Authentizität des Erhaltenen (bzgl. P) überprüfen kann.

Voraussetzung hierfür ist allerdings, daß die drei Prozessoren P, Q, R vor der Weitergabe von N gemeinsam ein Vorbereitungsprotokoll mit zuverlässigem Verteilkanal ausführen, zu dessen Ausführung die Nachricht N den Prozessoren P und Q bereits *bekannt* ist. Zudem erfordern authentisierte Verteilungsprotokolle meist, daß eine authentisierte Nachricht über bis zu t Prozessoren weitergegeben werden kann, nicht nur über einen.

Authentikationsnachweise sind daher für authentisierte Verteilungsprotokolle im allgemeinen ungeeignet. Indirekt werden sie jedoch die Grundlage der Verteilungsprotokolle aus Abschnitt 2.2 sein. Sie sind daher ausführlich in Abschnitt 2.1.2.2 beschrieben.

Einfache Pseudosignaturen (*unconditionally secure digital signatures*) erlauben wie Authentikationsnachweise die authentisierte Weitergabe einer Nachricht über genau einen Prozessor, etwa von P über Q zu R, und setzen ein Vorbereitungsprotokoll voraus, an dem alle drei Prozessoren teilnehmen [ChRo_91]. Für das Vorbereitungsprotokoll muß hier die weiterzugebende Nachricht jedoch noch nicht bekannt sein.

Auch einfache Pseudosignaturen sind zur Implementierung authentisierter Verteilungsprotokolle ungeeignet.

In Abschnitt 3.1 werden die einfachen Pseudosignaturen aber so zu (vollständigen) **Pseudosignaturen** erweitert, daß eine Weitergabe über eine beliebige *beschränkte* Anzahl an Prozessoren, etwa t, möglich ist. Dies erfüllt zwar nicht ganz die obige Forderung der Weitergebbarkeit, wird aber in Abschnitt 3.2 zur Implementierung des authentisierten Verteilungsprotokolls aus [DoSt_83] genügen.

Alle hier genannten Mechanismen mit Ausnahme des ersten (aus [SrTo_87]) sind nur im probabilistischen Sinne sicher: eine mit ihnen vorgenommene Authentikation ist mit einer zwar beliebig (exponentiell) kleinen, aber stets positiven Fehlerwahrscheinlichkeit unsicher.[2,3]

Fehlerfreie, für authentisierte Verteilungsprotokolle geeignete Authentikationsmechanismen für $t \geq n/3$ sind wegen der o.g. Unmöglichkeit fehlerfreier Verteilungsprotokolle für $t \geq n/3$ unmöglich.

1.1.2.2.2 Digitale Signaturen im kryptographischen Fehlfunktionsmodell

Mangels geeigneter Authentikationsmechanismen im byzantinischen Fehlfunktionsmodell werden für $t \geq n/3$ in authentisierten Verteilungsprotokollen **digitale Signaturen** verwendet [ChDw_89, PeSL_80].

Ein digitales Signatursystem stellt drei effizient zu berechnende Algorithmen *Gener*, *Sign* und *Test* bereit. Mit *Sign* werden Nachrichten signiert, d.h. authentisiert. Mit *Test* wird die Gültigkeit von Signaturen überprüft. Mittels des probabilistischen Algorithmus *Gener* kann jeder Prozessor ein Schlüsselpaar (SSi, STe)

---

[2] Ein Mechanismus hat eine *beliebig kleine Fehlerwahrscheinlichkeit*, wenn er durch einen *Sicherheitsparameter* σ parametrisiert ist und es für jedes $\varepsilon > 0$ einen Wert σ gibt, so daß der Mechanismus mit Sicherheitsparameter σ eine Fehlerwahrscheinlichkeit $\leq \varepsilon$ hat. Ist σ logarithmisch in $1/\varepsilon$ und geht nur polynomial in den Aufwand des Mechanismus ein, so ist die Fehlerwahrscheinlichkeit *exponentiell klein*.

[3] Erhält Q eine von P authentisierte Nachricht, so prüft Q in allen drei Mechanismen die Authentikation ausschließlich anhand der erhaltenen Information nach, und mit Wahrscheinlichkeit > 0 kann ein Störer diese Information *erraten*.

erzeugen, so daß gilt [DiHe_76, formal und ausführlich: GoMR_88]:[4]

Korrektheit Für alle Nachrichten N und alle möglichen Ausgaben (*Signaturen*) s von $Sign_{SSi}(N)$ gilt $Test_{STe}(N, s)$ = „ok".

Unfälschbarkeit Zu keiner Nachricht N kann ohne Kenntnis von SSi ein s' effizient berechnet werden mit $Test_{STe}(N, s')$ = „ok".

In einem Vorbereitungsprotokoll muß jeder Prozessor mittels *Gener* ein Schlüsselpaar (SSi, STe) wählen und den Testschlüssel *STe zuverlässig* an alle anderen Prozessoren verteilen (*public key*), während er den Signierschlüssel SSi geheim hält (*secret key*). Dies sichert die *Weitergebbarkeit* signierter Nachrichten.

Im byzantinischen Fehlfunktionsmodell kann es nach dieser Definition keine unfälschbaren digitalen Signaturen geben: Man kann zu einer beliebigen Nachricht N stets eine Signatur fälschen, indem man der Reihe nach alle möglichen Signaturen s durchprobiert, bis man eine gefunden hat, die $Test_{STe}(N, s)$ = „ok" erfüllt.

Nichtsdestotrotz gibt es Signaturen, die gemeinhin selbst gegenüber bösartigen Störern als sicher betrachtet werden. Ihre Unfälschbarkeit beruht darauf, daß das Problem, eine Fälschung zu finden, für Störer zu komplex ist. Das diesen Umstand beschreibende Fehlfunktionsmodell ist das **kryptographische**.

Betrachtet werden darin nur Signatursysteme mit einem **Sicherheitsparameter** k. Der Aufwand von *Gener*, *Sign* und *Test* ist jeweils polynomial in k. Grob gesagt postuliert das kryptographische Fehlfunktionsmodell, Störer könnten *keine* Berechnungen durchführen, die mehr als polynomialen Aufwand haben. Ein Signatursystem wird daher dann als **kryptographisch sicher** bezeichnet, wenn Signaturen in polynomialer Zeit nicht mit signifikant großer Wahrscheinlichkeit gefälscht werden können.[5]

Diese Formalisierung (die sehr ausführlich in [GoMR_88] dargestellt ist) hat allerdings zwei Nachteile:

Zum einen konnte in diesem Sinne noch *kein* Signatursystem als sicher bewiesen werden (dies würde $\mathcal{P} \neq \mathcal{NP}$ implizieren und gilt daher als recht schwierig). Bislang ist man daher schon zufrieden, wenn gezeigt ist, daß das Fälschen ebenso schwierig ist wie das Lösen eines als praktisch unlösbar betrachteten Problems. Die Annahme, ein bestimmtes Problem sei polynomial unlösbar, wird

---

4 Die Parameter *SSi* und *STe* für *Sign* und *Test* werden als Index geschrieben.

5 Etwas konkreter heißt dies [GoMR_88]: Die Wahrscheinlichkeit, eine Fälschung zu finden, ist für jedes in k polynomiale Fälschungsverfahren, jedes Polynom p und alle hinreichend großen k kleiner als $1/p(k)$. Die betrachteten Fälschungsverfahren können sich zum Finden einer Fälschung in k polynomial viele von ihnen gewählte Nachrichten signieren lassen und sind bereits dann erfolgreich, wenn sie zu irgendeiner neuen Nachricht eine gültige Signatur finden.

als **kryptographische Annahme** bezeichnet.

Für das Signatursystem aus [GoMR_88] kann man z. B. zeigen: Kann man in in k polynomialer Zeit Signaturen fälschen, so kann man auch in in k polynomialer Zeit BLUM-Zahlen faktorisieren, d. h. natürliche Zahlen, die aus zwei Primfaktoren bestimmter Gestalt und der Länge k bestehen [GoMR_88]. Trotz der langen mathematischen Geschichte des Faktorisierungsproblems ist bislang *kein* polynomialer Faktorisierungsalgorithmus bekannt (allerdings auch kein Beweis, daß ein solcher nicht existiert), und folglich ist bislang auch *kein* polynomialer Fälschungsalgorithmus für das Signatursystem aus [GoMR_88] bekannt.

Zum anderen wird nur die in k „asymptotische" Unfälschbarkeit bewiesen: Man zeigt bestenfalls, daß der Aufwand zum Fälschen in k schneller wächst als jedes Polynom in k. Für den praktischen Einsatz hilft dies jedoch nicht unbedingt viel: Wählt man einen konkreten Sicherheitsparameter und dazu passende konkrete Schlüssel (SSi, STe), so interessiert nur noch der Aufwand zur Lösung des entsprechenden konkreten Problems (also z. B. das Problem, eine ganz bestimmte Zahl zu faktorisieren).

In gewisser Hinsicht sicherer als die üblichen digitalen Signaturen sind die im folgenden Abschnitt 1.1.2.3 erwähnten *Fail-stop-Signaturen*, und die in Abschnitt 3.1 beschriebenen *Pseudosignaturen* werden für viele Anwendungen einen im byzantinischen Fehlfunktionsmodell sicheren Ersatz für digitale Signaturen darstellen.

1.1.2.3 Kombination von nichtauthentisierten und authentisierten Verteilungsprotokollen

Ein **kombiniert sicheres Verteilungsprotokoll** kombiniert die Fehlertoleranz eines authentisierten und eines nichtauthentisierten Verteilungsprotokolls:[6]

In [PfWa1_91, WaPf_89] wird ein authentisiertes Verteilungsprotokoll zur Tolerierung von $t < n$ (aus [DoSt_83: Th. 3]) mit einem beliebigen Verteilungsprotokoll Π_{byz} für das byzantinische Fehlfunktionsmodell kombiniert. Trifft das kryptographische Fehlfunktionsmodell zu, so toleriert das kombinierte Protokoll jedes $t < n$. Falls nicht, so toleriert es ebensoviele Störer wie Π_{byz} im byzantinischen Fehlfunktionsmodell.

Das kombiniert sichere Verteilungsprotokoll hat eine exponentiell kleine Fehlerwahrscheinlichkeit: es terminiert stets, aber mit exponentiell kleiner Wahrscheinlichkeit ist das Ergebnis falsch. (Dies gilt unabhängig davon, ob das

---

6 In [PfWa1_91, WaPf_89] wurde die Bezeichnung „adaptives byzantinisches Übereinstimmungsprotokoll" verwendet. Da mit „adaptiv" auch Protokolle bezeichnet werden, deren Aufwand stark von der aktuellen Fehlersituation abhängt [Baum_91, Baum1_91], wurde zur Vermeidung von Mißverständnissen die ursprüngliche Bezeichnung hier vermieden.

kryptographische Fehlfunktionsmodell zutrifft oder nicht.)

Grundlage für dieses kombiniert sichere Verteilungsprotokoll ist ein **Fail-stop-Signatursystem** [BlPW_91, HePe_92, HePP_92, Pfit_89, PfWa_90, PfWa_91, PfWa2_91, WaPf_89]: Dies ist ein digitales Signatursystem, das in einem kryptographischen Fehlfunktionsmodell im üblichen Sinne fälschungssicher ist[7], aber die zusätzliche Eigenschaft hat, daß ein Prozessor, dessen Signatur dennoch gefälscht wurde, allen anderen Prozessoren die Unsicherheit des Systems nachweisen kann. Die Möglichkeit, gegebenenfalls die Unsicherheit des Systems nachweisen zu können, hängt von keinen kryptographischen Annahmen ab, ist jedoch mit einer exponentiell kleinen Fehlerwahrscheinlichkeit behaftet.

1.1.2.4 Schwächere Fehlfunktionsmodelle

Der Vollständigkeit halber sei erwähnt, daß neben dem hier verwendeten byzantinischen und dem in Abschnitt 1.1.2.2 genannten kryptographischen noch weitere Fehlfunktionsmodelle üblich sind.

Das praktisch relevanteste ist das **Anhaltemodell**, das besagt, daß nur **Anhalteausfälle** (*fail-stop, crash*) auftreten [Echt_90, ScSc_83]: Ein Störer verhält sich bis zu einem bestimmten Zeitpunkt völlig korrekt (d.h. sendet nur erlaubte Nachrichten), und interagiert nach diesem Zeitpunkt nicht mehr mit anderen Prozessoren. Er fällt also vollständig aus. Da in diesem Modell authentisierte Protokolle mit trivialen Authentikationsmechanismen einsetzbar sind, können beliebig viele Anhalteausfälle toleriert werden. Das Anhaltemodell ist im Gegensatz zum byzantinischen Fehlfunktionsmodell ausschließlich an *technischen* Störungen orientiert, die gemeinhin gewissen stochastischen Gesetzmäßigkeiten unterworfen sind.

Im folgenden wird nur das Maximum an Fehlertoleranz betrachtet, so daß Abschwächungen des byzantinischen Fehlfunktionsmodells nicht betrachtet werden. Für eine ausführlichere Diskussion schwächerer Fehlfunktionsmodelle sei auf [Echt_90] verwiesen.

1.2 Verteiltes System mit Fehlern

Im folgenden soll das für den Rest der Arbeit verwendete *Modell* eines verteilten Systems etwas genauer beschrieben werden. Abschnitt 1.2.1 beschreibt zunächst das korrekte System, Abschnitt 1.2.2 das byzantinisch fehlerhafte. In Abschnitt 1.2.3 werden Qualitätsmerkmale eines Protokolls beschrieben.

---

[7] Als kryptographische Annahme kann z.B. die Annahme dienen, BLUM-Zahlen seien in polynomialer Zeit nicht faktorisierbar [GoMR_88].

1.2.1 Das korrekte System

Ein **Prozessor** ist ein probabilistischer Automat mit unendlichem Zustandsraum (z. B. $\mathbb{N}$), der als Ein- bzw. Ausgaben Nachrichten empfangen bzw. senden kann. Die n Prozessoren werden mit $P_1, \ldots, P_n$ bezeichnet, als Variablen werden $P_i, P_j, \ldots$ und P, Q, R, S, T verwendet.

Der aktuelle Zustand eines Prozessors und die Liste der bislang empfangenen Nachrichten wird als **Situation** bezeichnet.

In einer **Runde** wählt ein Prozessor entsprechend einer vorgegebenen Wahrscheinlichkeitsverteilung in Abhängigkeit von seiner Situation eine **Reaktion**, d. h. einen neuen internen Zustand und eine Liste von zu sendenden Nachrichten (formaler ist dies in [BeGW_88, BrDo_84, GrYa_89] beschrieben).

Welche Reaktionsmöglichkeiten für einen Prozessor vorgesehen sind, wird durch sein **Prozessorprotokoll** beschrieben. Sieht dieses für jede Situation genau eine mögliche Reaktion vor, so heißt es **deterministisch**, ansonsten **probabilistisch**. Letzteres heißt dann, daß das Verhalten des Prozessors durch Wahrscheinlichkeitsverteilungen (interne „Münzwürfe“) beschrieben wird.

Alle korrekten Prozessoren arbeiten **synchron**, d. h. beginnen und beenden ihre Runden gleichzeitig.

Die Prozessoren können jeweils paarweise miteinander **kommunizieren**.

Auch die Kommunikation erfolgt **synchron**: Gibt ein Prozessor P_i in der r-ten Runde das Paar (N,j) aus, so erhält P_j vor der $(r+1)$-ten Runde (d. h. als Eingabe für diese Runde) das Tupel (i,N,j). i wird **Absender**, j **Adressat** der Nachricht N genannt.

Ein **Protokoll** besteht aus einem Prozessorprotokoll für jeden Prozessor.

Zu Beginn einer Protokollausführung befindet sich jeder Prozessor P_i in einem **Startzustand**, der die lokalen Eingaben des Prozessors (z. B. den Index i) enthält. Für die byzantinische Verteilung enthält der Startzustand des korrekten Transmitters zusätzlich den zu verteilenden Wert.

Ein Prozessor beendet das Protokoll, wenn er in einen **Stopzustand** übergeht. Das Protokoll **terminiert**, wenn alle Prozessoren einen Stopzustand erreicht haben.[8]

Für ein Verteilungsprotokoll beschreibt ein **akzeptierender** Zustand Z von P eindeutig einen Wert $v(Z)$ als denjenigen, den Prozessor P endgültig als den Wert des Transmitters akzeptiert. Für alle möglichen Folgezustände Z' von Z

---

[8] Im folgenden werden nur solche Protokolle betrachtet, die nach einer begrenzten Anzahl an Runden terminieren.

gilt: Z' ist ebenfalls akzeptierend, und $v(Z') = v(Z)$.[9]

Führt im folgenden ein Prozessor zwei Protokolle Π_1, Π_2 *hintereinander* aus, so müssen die relevanten Teile des Stopzustands von Π_1 in den Startzustand von Π_2 übernommen werden.

Dies wird wie folgt modelliert: Erreicht der Prozessor den Stopzustand von Π_1, so sendet er eine Nachricht N an sich selbst, die alle relevanten Daten enthält (also z. B. globale Eingaben, Ergebnisse von Π_1). Der Stopzustand von Π_1 wird dann durch einen Startzustand von Π_2 ersetzt, in dem der Prozessor N empfängt und verarbeitet.

Das Senden dieser Nachricht wird im folgenden nicht explizit betrachtet.

Ein konkretes Prozessorprotokoll wird durch eine **probabilistische Turing-Maschine** angegeben.[10]

Eine probabilistische Turing-Maschine ist eine Mehrband-Turing-Maschine mit einem speziellen *Zufallsband*, von dem nur in einer Richtung gelesen und das nicht beschrieben werden kann (vgl. z. B. [Gill_77]). Das Lesen vom Zufallsband entspricht dem Werfen einer fairen Münze: liest die Maschine ein Zeichen z, so tritt $z = 0$ und $z = 1$ jeweils mit Wahrscheinlichkeit $1/2$ auf.

Der Zustand eines Prozessors besteht dann aus dem Programmzustand seiner Turing-Maschine, den bislang von ihm beschriebenen oder gelesenen Teilen seiner Bänder (einschließlich Zufallsband) und seinen lokalen Eingaben (d. h. des beschriebenen Teils seines Eingabebandes).

Setzt man die Turing-Maschinen direkt in das obige Zustandsübergangsmodell ein, so erhält man die zur Modellierung kryptographischer Protokolle mittlerweile üblichen interaktiven Turing-Maschinen [BeKP_86, FeMi_88, GoMR_89].

Im folgenden werden Prozessorprotokolle nur informell beschrieben. Alle Protokolle werden in dem Sinne uniform sein, daß sie für jede Anzahl n von Prozessoren verwendet werden können.

Probabilistische Protokolle können gegenüber deterministischen zwei Vorteile haben.

Zum einen kann für eine feste Problemstellung der Lösungsaufwand eines probabilistischen Protokolls streuen, und der *Erwartungswert* des Aufwands kann geringer sein als der Aufwand eines deterministischen.

Zum anderen sind manche Probleme deterministisch überhaupt nicht lösbar, wohl aber „näherungsweise“ durch probabilistische Protokolle: Die Fehlerwahr-

---

9 Akzeptierende Zustände sind nicht notwendigerweise auch Stopzustände, denn die korrekten Prozessoren müssen sich nicht notwendigerweise alle gleichzeitig für einen Endwert entscheiden. Dies wird im folgenden jedoch nur in Protokoll 2.9 der Fall sein.

10 Für deterministische Protokolle genügen natürlich auch deterministische Turing-Maschinen.

scheinlichkeit ε, daß das Protokoll kein korrektes Ergebnis liefert, ist zwar echt größer Null, kann aber durch entsprechend großen Aufwand beliebig klein werden.[11]

Für beide Vorteile ist byzantinische Verteilung ein Beispiel:

- Für $t < n/3$ benötigen deterministische Protokolle wenigstens $t+1$ Runden [FiLy_82], während probabilistische mit erwartet konstanter Rundenanzahl zum Ziel gelangen [FeMi_88, GoPe_90].
- Für $t \geq n/3$ existieren keine deterministischen Lösungen [PeSL_80], wohl aber, wie im folgenden gezeigt wird, probabilistische mit exponentiell kleinem Fehler.

1.2.2 Das byzantinische Fehlermodell

Ein **Fehlermodell** beschreibt, wie fehlerhaft ein Gesamtsystem werden darf, ohne daß hierdurch die Erbringung der spezifizierten Funktion des Systems verhindert wird. Das **byzantinische Fehlermodell** besteht aus dem *byzantinischen Fehlfunktionsmodell* und einer Reihe von Annahmen, die in Abschnitt 1.1 schon kurz genannt wurden und hier ausführlicher beschrieben werden.

Das Fehlermodell wird hier stets die maximale Anzahl an fehlerhaften Prozessoren beschränken: Es wird angenommen, bis zu t der n Prozessoren können fehlerhaft, d. h. zu **Störern**, werden.

Das **byzantinische Fehlfunktionsmodell** macht *keine* Annahmen über das Verhalten von Störern, d. h. statt des korrekten Prozessorprotokolls kann ein Störer ein beliebig davon abweichendes verwenden [ChDw_89, PeSL_80].[12]

Im schlimmsten Fall sind alle Störer durch einen intelligenten „Saboteur" koordiniert, für den zudem die Komplexität eines Problems ohne Belang ist. Formal genügt es daher, alle Störer zusammen durch ein **Störprotokoll** darzustellen. Dieses erhält die Eingaben und liefert die Ausgaben aller Störer.

Eine **Fehlersituation** wird beschrieben durch die Angabe, welche Prozessoren fehlerhaft sind und welches Störprotokoll sie verfolgen. Eine Fehlersituation heißt **zulässig**, wenn in ihr höchstens t Prozessoren fehlerhaft sind.

Die internen Zustände, Zufallsbänder und lokalen Eingaben der korrekten Prozessoren sind vor den Störern verborgen.[13]

---

[11] Ein Protokoll dieser Art nennt man oft ein „Monte-Carlo-Protokoll". Kann man dem Ergebnis ansehen, ob es korrekt ist, so nennt man es oft ein „Las-Vegas-Protokoll" (nach [BrBr_88]).

[12] Das Störprotokoll muß nicht einmal durch eine probabilistische Turing-Maschine berechenbar sein. Praktisch ist dies jedoch bedeutungslos.

[13] Manche Protokolle, etwa das Verteilungsprotokoll für $t < n/3$ aus [PeSL_80], kommen ohne diese Annahme aus. Für probabilistische Protokolle ist diese Annahme jedoch meist notwendig.

Störer haben hinsichtlich der Kommunikation der korrekten Prozessoren nur sehr geringe Möglichkeiten: Die Verbindung von einem korrekten Prozessor P zu einem korrekten Prozessor Q ist **sicher**, d. h. hat folgende Eigenschaften:

Integrität Sendet P in der r-ten Runde eine Nachricht N an Q, so kommt diese unverändert bei Q vor Beginn der (r+1)-ten Runde an.

Authentizität Empfängt Q eine Nachricht N von der Verbindung zu P, so ist Q sicher, daß N wirklich von P gesendet wurde.

Störer können die Kommunikation zwischen korrekten Prozessoren also nicht beeinflussen, und Verbindungen zwischen korrekten Prozessoren werden nicht fehlerhaft. (Wird die Verbindung zwischen P und Q real doch fehlerhaft, so wird im Modell P oder Q fehlerhaft.) Im *Modell* heißt dies einfach, daß das Störprotokoll auf die Zuordnung von gesendeten zu empfangenen Nachrichten keinen Einfluß hat.

Die Annahme der Existenz sicherer Verbindungen wird **Zusammenhangsannahme** genannt.

Sichere Verbindungen können jedoch abhörbar sein. Für manche Teile der folgenden Protokolle ist die Abhörbarkeit von Verbindungen jedoch nicht tolerierbar, d. h. es werden **sichere nichtabhörbare Verbindungen** benötigt. Diese haben neben den beiden schon genannten noch die folgende Eigenschaft:

Konzelation Sind P, Q korrekte Prozessoren und sendet P eine Nachricht N an Q, so erhält kein Beobachter Information über N.

Im *Modell* wird die Abhörbarkeit einer Verbindung sehr einfach dargestellt: Ist die Verbindung zwischen P_i und P_j abhörbar, so erhält im Modell das Störprotokoll alle Nachrichten N, die P_i an P_j sendet, als Eingabe (i,N,j). Ist die Verbindung nichtabhörbar, so geschieht dies nicht.

Die Integrität einer Verbindung muß physikalisch gewährleistet sein, Authentizität und Konzelation können unter gewissen Annahmen aus der Integrität abgeleitet werden (vgl. Abschnitt 1.4.2).

Für die byzantinische Verteilung werden sichere bzw. sichere nichtabhörbare Verbindungen genügen. Bevor jedoch eines der folgenden byzantinischen Verteilungsprotokolle ausgeführt werden kann, muß ein **Vorbereitungsprotokoll** ausgeführt werden. Dies kann beliebige Zeit vor dem eigentlichen Protokoll geschehen, insbesondere also bevor der Wert des Transmitters festgelegt ist.

Während der Ausführung des Vorbereitungsprotokolls wird neben sicheren nichtabhörbaren Verbindungen ein **zuverlässiger Verteilkanal** vorausgesetzt. Sendet P in der r-ten Runde eine Nachricht über einen solchen Kanal, so gilt für alle korrekten Prozessoren Q, R:

Terminierung Q empfängt vor Beginn der (r+1)-ten Runde eine Nachricht.

Korrektheit Ist P korrekt, so empfängt Q die Nachricht, die P verteilen wollte.

Konsistenz Q und R empfangen in jedem Fall dieselbe Nachricht.

Authentizität Empfängt Q eine Nachricht N über den Verteilkanal, so kann Q den Sender von N zweifelsfrei feststellen.

Möchte im *Modell* P_i eine Nachricht N zuverlässig verteilen, so gibt er das Paar (N,*) aus. Für die nächste Runde erhalten dann alle Prozessoren die Eingabe (i,N,*).[14]

Die Störer können die **Synchronisation** der Uhren korrekter Prozessoren und der Kommunikation zwischen korrekten Prozessoren nicht beeinflussen.

Dies heißt jedoch *nicht*, daß die Störer sich an die vorgegebene synchrone Rundeneinteilung halten müssen: Im schlimmsten Fall erhalten sie alle Nachrichten, die die korrekten Prozessoren in einer Runde senden, noch bevor sie selbst ihre Nachrichten für diese Runde senden.

Im Modell wird daher jede Runde künstlich in zwei Teilrunden aufgespalten: In der ersten Teilrunde wählen alle *korrekten* Prozessoren ihre Reaktion aus, in der zweiten dann das *Störprotokoll*, wobei es als Eingabe bereits die für die Störer zugänglichen Nachrichten der korrekten Prozessoren aus der ersten Teilrunde (d. h. derselben Runde) erhält.[15]

1.2.3 Qualität eines Protokolls

Wesentlich zur Beschreibung der Qualität eines Protokolls sind seine Fehlertoleranz und sein Aufwand.

Die **Fehlertoleranz** wird durch das angenommene Fehlermodell und die verbliebene Fehlerwahrscheinlichkeit beschrieben.

Das unterstellte **byzantinische Fehlermodell** ist in Abschnitt 1.2.2 beschrieben. Lediglich die Anzahl t ist variabel.[16]

Zu einer gegebenen Zahl R von Runden und einer zulässigen Fehlersituation sei $W_{Fehler}(R)$ die Wahrscheinlichkeit, mit der R Runden nach Beginn des

---

[14] Mit einem zuverlässigen Verteilkanal während der Hauptphase könnte das byzantinische Verteilungsproblem trivial gelöst werden. Ist umgekehrt allen Prozessoren der Sender P und der Sendezeitpunkt bekannt, so kann ein byzantinisches Verteilungsprotokoll dazu verwendet werden, einen zuverlässigen Verteilkanal zu realisieren.

[15] In [GrYa_89] wird dies das *sequentielle Modell* genannt, im Gegensatz zum *simultanen Modell*, in dem auch die Störer die Ausgaben der korrekten Prozessoren erst in der nächsten Runde erhalten.

[16] Sind anfangs alle Prozessoren korrekt und kann ein stochastisches Modell dafür angegeben werden, wie korrekte Prozessoren zu Störern werden, so muß nur die Fehlerwahrscheinlichkeit betrachtet werden [Baba_87].

Protokolls das zu lösende Problem *nicht* gelöst wurde (d.h. das Protokoll ist (noch) nicht terminiert, oder die gefundene Lösung ist fehlerhaft). W_{Fehler} ist definiert über das probabilistische Verhalten aller Prozessoren, die fehlerhaften mit eingeschlossen. Die **Fehlerwahrscheinlichkeit** $\varepsilon(R)$ eines Protokolls ist dann eine *obere Schranke* für $W_{Fehler}(R)$ für alle zulässigen Fehlersituationen.

Hat also ein Protokoll die Fehlerwahrscheinlichkeit $\varepsilon(R)$, so löst es für *alle* zulässigen Fehlersituationen aus bis zu t Störern das byzantinische Verteilungsproblem mit Wahrscheinlichkeit $\geq 1-\varepsilon(R)$ innerhalb von R Runden.

Der **Aufwand** eines Protokolls besteht im wesentlichen im Rechenaufwand der Prozessoren und im Kommunikationsaufwand.

Zunächst betrachte man *einen* konkreten Protokollablauf bis zur Terminierung: Da Prozessorprotokolle Turing-Maschinen sind, sei der *Rechenaufwand eines Prozessors* die Anzahl der Operationen dieser Turing-Maschine in diesem Ablauf, und der **Rechenaufwand** des Protokollablaufs sei das Maximum der Rechenaufwände aller korrekten Prozessoren.[17] Für den Kommunikationsaufwand sind zwei Maße üblich: Die **Rundenanzahl** des Protokollablaufs (*distributed complexity*) gibt an, wieviele Runden ausgeführt wurden. Der **Nachrichtenaufwand** des Protokollablaufs (*communication complexity*) gibt an, wieviele Nachrichten bzw. Bits alle korrekten Prozessoren zusammen sendeten, wobei zwischen über einen zuverlässigen Verteilkanal (**Verteilaufwand**) und über sichere Verbindungen gesendeten Nachrichten unterschieden wird.

Ein Protokollablauf ist im allgemeinen nicht nur durch die Startzustände der korrekten Prozessoren bestimmt (d.h. durch ihre Eingaben), sondern auch durch die Fehlersituation (welche Prozessoren sind wie fehlerhaft?) und durch die Zufallsentscheidungen *aller* Prozessoren. Als Aufwand kann daher einer solchen Anfangssituation nicht ein fester Wert, sondern muß eine Zufallsvariable zugeordnet werden. Für diese sei wie üblich der *maximale* bzw. *erwartete Aufwand* definiert.

Der Aufwand des Protokolls wird durch die ungünstigsten Anfangssituationen bestimmt. Daher sei der **maximale** (bzw. **erwartete**) **Aufwand eines Protokolls** das Maximum des maximalen (bzw. erwarteten) Aufwands, gebildet über alle gemäß Fehlermodell zulässigen Anfangssituationen. Für alle folgenden Protokolle wird der maximale (bzw. erwartete) Aufwand endlich sein.

Die Fehlerwahrscheinlichkeit und der Aufwand eines Protokolls werden in dieser Arbeit meist von n, t und einem frei wählbaren **Sicherheitsparameter** $\sigma \in \mathbb{N}$ abhängen. Je größer σ ist, desto kleiner ist die Fehlerwahrscheinlichkeit und

---

[17] Im folgenden wird der Rechenaufwand stets nur soweit betrachtet, bis klar ist, daß er deterministisch polynomial ist. Auf genauere Eigenschaften der Turing-Maschinen (z.B. Anzahl und Art der Bänder) kommt es daher nicht an.

desto größer der Aufwand.

Alle folgenden Protokolle werden einen in n, t und σ polynomialen Aufwand haben, ihre Fehlerwahrscheinlichkeit ist meist $\leq 2^{-\sigma}$, also im üblichen Sinne exponentiell klein.

Ein Ereignis, das mit exponentiell kleiner Wahrscheinlichkeit eintritt, heißt **fast unmöglich**, eines, das mit Wahrscheinlichkeit exponentiell nahe 1 eintritt, **fast sicher**.

1.3 Bekannte Grenzen der byzantinischen Verteilung

Für das Problem der byzantinischen Verteilung sind zahlreiche Verfahren und einige wenige Grenzen bekannt (vgl. auch Abschnitt 1.1.2).

Die bekannteste Grenze für byzantinische Verteilung enthält sicher der folgende, in Abschnitt 1.1.2.1 schon erwähnte, Satz aus [PeSL_80]:

Satz 1.1: Fehlerfrei kann byzantinische Verteilung höchstens dann erreicht werden, wenn $t < n/3$ ist, d. h. wenn in jedem Fall mehr als zwei Drittel aller Prozessoren korrekt arbeiten. ♦

Beweisidee (nach [LaSP_82]): Man betrachtet zunächst den einfachsten Fall und nimmt an, für $n = 3$, $t = 1$ gäbe es eine fehlerfreie Lösung. Seien P, Q, R die drei Prozessoren, P sei der Transmitter, und o. B. d. A. sei der Anfangswert des Transmitters aus {0, 1}. Man betrachte folgende drei Situationen (Bild 1-1):

1. *Situation:* Nur R ist fehlerhaft. P möchte „0“ verteilen, R verhält sich gegenüber Q aber genau so, wie wenn P den Wert „1“ verteilen würde. Q muß sich für „0“ entscheiden.
2. *Situation:* Nur Q ist fehlerhaft. P möchte „1“ verteilen, Q verhält sich aber gegenüber R genau so, wie wenn P den Wert „0“ verteilen würde. R muß sich für „1“ entscheiden.
3. *Situation:* Nur P ist fehlerhaft und verhält sich gegenüber Q wie in Situation 1, gegenüber R wie in Situation 2. Gegeneinander verhalten sich die korrekten Prozessoren Q und R daher wie in Situation 1 bzw. Situation 2. Da Q und R die Situation 1 bzw. 2 nicht von 3 unterscheiden können, muß sich Q für „0“ und R für „1“ entscheiden. Es herrscht keine Konsistenz!

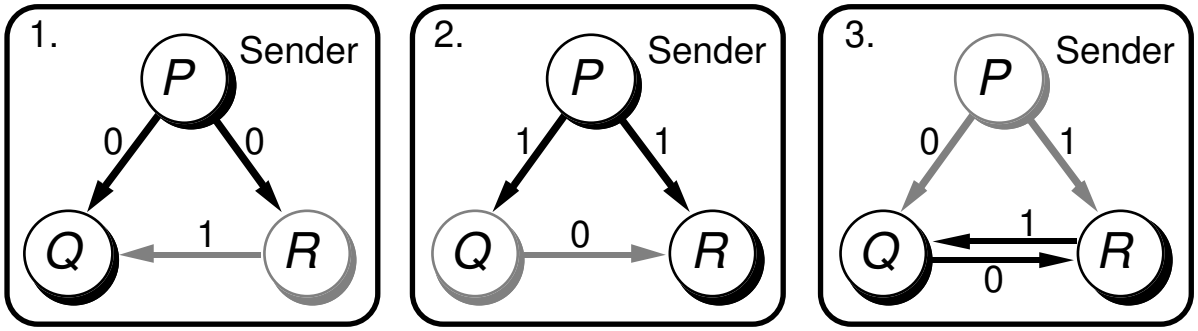


Bild 1-1 Beweisskizze für Satz 1.1. Störer sind grau, korrekte Prozessoren schwarz gezeichnet. Ein Pfeil von *X* nach *Y* mit der Aufschrift *b* bedeutet, daß sich *X* gegenüber *Y* so verhält, wie wenn der Transmitter *P* den Anfangswert *b* hätte.

Der Fall „n beliebig und $t \geq n/3$“ wird auf diesen einfachsten Fall zurückgeführt:

Angenommen, Π sei ein Protokoll, das für $n = 3t$ fehlerfrei bis zu t Störer toleriert. Dann würde folgendes Protokoll für 3 Prozessoren einen Störer tolerieren: Aus den n Prozessorprotokollen von Π bilde man drei disjunkte Mengen $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{R}$ mit jeweils t Elementen. $\mathcal{P}$ enthalte das Protokoll des Transmitters. Prozessor P (bzw. Q, R) simuliere alle Prozessoren in $\mathcal{P}$ (bzw. $\mathcal{Q}$, $\mathcal{R}$), d.h. sende und empfange alle Nachrichten der entsprechenden t simulierten Prozessoren. Da höchstens ein realer und damit höchstens t simulierte Prozessoren fehlerhaft werden, findet das simulierte Protokoll die fehlerfreie Lösung.

Da dies im Widerspruch zur eben gezeigten Unmöglichkeit eines Protokolls für $n = 3$, $t = 1$ steht, kann ein solches Protokoll Π nicht existieren. ❒

Die Grenze für t aus Satz 1.1 kann auch tatsächlich erreicht werden.

Bereits das erste überhaupt veröffentlichte, deterministische Verteilungsprotokoll aus [LaSP_82, PeSL_80] toleriert jedes $t < n/3$ in $t+1$ Runden, allerdings mit exponentiellem Nachrichtenaufwand. Die Anzahl von $t+1$ Runden ist für deterministische Protokolle optimal [FiLy_82].[18] Mit polynomialem Nachrichtenaufwand ($O(nt+t^3 \log(t))$ bit) wird in [DFFL_82] jedes $t < n/3$ in $2t+3$ Runden toleriert, in [MoWa_88] jedes $t < n/8$ in $t+1$ Runden. Nach [DoRe_85] ist $\Omega(nt)$ eine untere Schranke für die Anzahl an zu sendenden Nachrichten.

Probabilistische Verteilungsprotokolle können ihr Ziel in erwartet weniger als $t+1$ Runden erreichen [Brac_85, BrTo_85, ChDw_89, Rabi_83]. Mit polynomialem Nachrichtenaufwand und erwartet *konstanter* Rundenanzahl wird in [FeMi_88] jedes $t < n/3$ toleriert. Wie alle bis dahin bekannten „schnellen“ probabilistischen Protokolle hat dieses Protokoll jedoch einen Fehler: nach jeder vorgegebenen Rundenanzahl ist die Wahrscheinlichkeit, daß es noch nicht terminiert ist, echt

---

[18] Dies gilt auch für deterministische authentisierte Verteilungsprotokolle [DeLM_82, DoSt_83], nach [Hadz_84] sogar für das Anhaltemodell. Siehe auch [DwMo_90].

größer Null. Es läßt sich aber so mit einem in O(t) Runden terminierenden deterministischen Verteilungsprotokoll kombinieren, daß die Kombination in erwartet konstanter und maximal O(t) Rundenanzahl terminiert [GoPe_90]. (Die Konstruktion aus [GoPe_90] kann auch auf andere probabilistische Verteilungsprotokolle angewandt werden.)

Durch die genannten Protokolle kann byzantinische Verteilung also fehlerfrei für jedes $t < n/3$ in O(t) Runden und mit polynomialem Nachrichtenaufwand realisiert werden. Während der eigentlichen Protokollausführung werden sichere Verbindungen vorausgesetzt. Das Protokoll aus [FeMi_88] setzt sogar sichere nichtabhörbare Verbindungen voraus. Ein Vorbereitungsprotokoll mit zuverlässiger Verteilung wird nur dazu benötigt, das Protokoll zu installieren und die Rollen der einzelnen Prozessoren und den Beginn der Protokollausführung festzulegen.

Nach Satz 1.1 muß byzantinische Verteilung für $t \geq n/3$ notwendigerweise fehlerbehaftet sein.

Die Größe der Fehlerwahrscheinlichkeit ε wird in [DoDw_91, GrYa_89] untersucht.[19] Für jedes Verteilungsprotokoll, das nur auf paarweiser Kommunikation zwischen den Prozessoren aufbaut, gilt: Stehen nur abhörbare Verbindungen zur Verfügung, so ist $\varepsilon \geq (3 - \sqrt{5})/2 > 0{,}381$ [GrYa_89], ansonsten $\varepsilon \geq 1/3$ [DoDw_91].

Die im folgenden vorgestellten Verteilungsprotokolle erreichen für $t \geq n/3$ eine *beliebig* kleine Fehlerwahrscheinlichkeit ε, insbesondere $\varepsilon < 1/3$. Aus den oben genannten Unwahrscheinlichkeitsaussagen folgt, daß dies ohne zusätzliche Annahmen unmöglich ist.

Im folgenden wird daher angenommen, daß vor dem Verteilungsprotokoll ein Vorbereitungsprotokoll ausgeführt wird, für das ein zuverlässiger Verteilkanal und nichtabhörbare Verbindungen zur Verfügung stehen (vgl. Abschnitt 1.2.2).

Auf keine der beiden Annahmen für das Vorbereitungsprotokoll kann verzichtet werden: Ohne zuverlässigen Verteilkanal kann das Vorbereitungs- als Teil des Verteilungsprotokolls betrachtet werden, so daß aus [DoDw_91] $\varepsilon \geq 1/3$ folgt. Ohne nichtabhörbare Verbindungen verändert das Vorbereitungsprotokoll das betrachtete Protokoll Π nur in einer für die Störer beobachtbaren Weise in ein neues Protokoll Π'. Auch für dieses Π' gilt nach [GrYa_89] $\varepsilon \geq (3 - \sqrt{5})/2$.

---

[19] Die Unwahrscheinlichkeit byzantinischer Verteilung für $t \geq n/3$ wurde nach [GrYa_89] erstmals von KARLIN und YAO untersucht, allerdings für ein etwas schwächeres Fehlermodell: Zusätzlich zur Abhörbarkeit von Verbindungen wurde angenommen, auch Störer könnten Nachrichten, die in der r-ten Runde gesendet wurden, erst in der (r+1)-ten Runde verarbeiten. Dann ist $\varepsilon \geq 1/3$ (zitiert nach [GrYa_89]).

1.4 Diskussion der Annahmen bezüglich Synchronisation und Kommunikation

Das byzantinische Fehlermodell enthält zwar kein einschränkendes Fehlfunktionsmodell, dafür aber einschränkende Annahmen über den Einfluß von Störern auf die Synchronisation und Kommunikation korrekter Prozessoren, und geht davon aus, daß eine exponentiell kleine Fehlerwahrscheinlichkeit nicht stört.

1.4.1 Synchronisation

Die korrekten Prozessoren arbeiten das Protokoll in Runden ab, d.h. arbeiten und kommunizieren synchron.

Hinreichend zur Realisierung dieser Forderung sind die folgenden drei Annahmen (in Anlehnung an [LaSP_82]). Sie gehen idealisierend von der Existenz einer realen Zeit aus, relativ zu der die Ungenauigkeit lokaler Uhren definiert werden kann. Die Zeitpunkte τ, σ beziehen sich auf diese reale Zeit.

Synchrone Uhren

Jeder korrekte reale Prozessor P_i besitzt eine lokale Uhr, d.h. einen Operator C_i, der zum Zeitpunkt τ gelesen die lokale Zeit $C_i(\tau)$ angibt. Für diese lokalen Uhren gilt:

- Es ist eine Konstante $\Delta_C \geq 0$ bekannt, so daß für alle korrekten P_i und P_j und alle Zeitpunkte τ gilt
$$|\, C_i(\tau) - C_j(\tau) \,| \leq \Delta_C.$$
- Es ist eine Konstante $\Delta_D \geq 0$ bekannt, so daß für alle korrekten P_i und alle Zeitpunkte $\tau > \sigma$ gilt
$$(1 + \Delta_D)^{-1} \leq \frac{C_i(\tau) - C_i(\sigma)}{\tau - \sigma} \leq (1 + \Delta_D)$$

Mit anderen Worten: die Abweichungen der lokalen Uhren untereinander und ihre Drift gegenüber der realen Zeit sind durch bekannte Konstanten beschränkt [HSSD_84]. ♦

Synchrone Kommunikation

Es ist eine Konstante $\Delta_T \geq 0$ bekannt, so daß für alle Prozessoren P, Q, alle Zeitpunkte τ und alle Nachrichten N aus einem endlichen Nachrichtenraum gilt: Hat P zum Zeitpunkt τ die Nachricht N an Q gesendet, so hat Q die Nachricht spätestens zum Zeitpunkt $\tau + \Delta_T$ empfangen. ♦

Begrenzte Verarbeitungszeit

> Es ist eine Konstante $\Delta_P \geq 0$ bekannt, so daß jeder korrekte Prozessor, der eine Runde zu einem lokalen Zeitpunkt T begonnen hat, spätestens zur lokalen Zeit $T + \Delta_P$ alle Nachrichten der letzten Runde eingelesen und alle Nachrichten für diese Runde erzeugt und gesendet hat. ♦

Unter diesen Annahmen kann die Rundeneinteilung leicht simuliert werden.

Sei T die lokale Zeit, zu der Prozessor P_i spätestens eine Nachricht an Prozessor P_j gesendet haben muß, und sei τ die T entsprechende reale Zeit (d.h. $C_i(\tau) = T$). Ist P_i korrekt, so kommt die Nachricht spätestens zur realen Zeit $\tau + \Delta_T$ bei P_j an. Aus dessen Sicht ist, wegen der beschränkten Drift,

$$C_j(\tau + \Delta_T) \leq C_j(\tau) + \Delta_T\,(1 + \Delta_D).$$

Wegen der beschränkten Abweichung der lokalen Uhren untereinander ist

$$C_j(\tau) \leq C_i(\tau) + \Delta_C = T + \Delta_C.$$

Kommt die Nachricht also nicht spätestens zur lokalen Zeit

$$T + \Delta_C + \Delta_T\,(1 + \Delta_D)$$

bei P_j an, so ist sicher, daß P_i sie nicht rechtzeitig gesendet hat, d.h. fehlerhaft ist.

Sei $T(r)$ die lokale Zeit, zu der *alle* Prozessoren die r-te Runde begonnen haben müssen. Dann haben alle Prozessoren spätestens zur lokalen Zeit

$$T(r{+}1) := T(r) + \Delta_P + \Delta_C + \Delta_T\,(1 + \Delta_D)$$

alle Nachrichten empfangen und können daher die $(r{+}1)$-te Runde beginnen.

Ist die maximale Abweichung $\Delta_C(\tau)$ der lokalen Uhren zu einem bestimmt Zeitpunkt τ bekannt, so kann man aus der Drift Δ_D leicht für jedes $\Delta_\tau > 0$ die maximale Abweichung $\Delta_C(\tau{+}\Delta_\tau)$ abschätzen [HSSD_84]: Es ist

$$\begin{aligned} |\,C_i(\tau{+}\Delta_\tau) - C_j(\tau{+}\Delta_\tau)\,| \;&=\; |\,(C_i(\tau) - C_j(\tau)) + (C_i(\tau{+}\Delta_\tau) - C_i(\tau)) - (C_j(\tau{+}\Delta_\tau) - C_j(\tau))\,| \\ &\leq \Delta_C(\tau) + 2\,\Delta_\tau\,(1{+}\Delta_D) =: \Delta_C(\tau{+}\Delta_\tau). \end{aligned}$$

Für endliche Zeiträume genügt es daher, wenn die Anfangsabweichung $\Delta_C(0)$ der lokalen Uhren und Δ_D bekannt sind.

Unter normalen Betriebsbedingungen sind die drei oben genannten Zeitbedingungen gerade die an ein Realzeitsystem, sollten also in der Praxis durchaus erfüllbar sein. In [HSSD_84] wird etwa $\Delta_D = 10^{-6}$ und in einem lokalen Netz $\Delta_T = 10^{-1}$ s (für „kurze“ Nachrichten) als realistisch bezeichnet.

Geht man allerdings von einem böswilligen Saboteur aus, d.h. sind die Betriebsbedingungen extrem ungünstig, so muß man letztendlich zu physikalischen Schutzmaßnahmen greifen.

Die Verabeitungszeit Δ_P hängt auch vom betrachteten Protokoll ab. Im folgenden werden nur Protokolle betrachtet, für die jede Runde nach bekannt beschränkter Zeit endet, d.h. für die Δ_P deterministisch beschränkt werden kann.

Die erste der drei Bedingungen kann unter Umständen etwas abgeschwächt werden: Existieren Δ_D, Δ_T und Δ_P und sind jeweils bekannt, so kann Δ_C, also die Abweichung der lokalen Uhren voneinander, durch Protokolle zur Uhrensynchronisation *konstant* beschränkt werden (eine *lineare* Beschränkung ist, wie oben gezeigt, stets möglich).

Sinnvollerweise muß man für die Uhrensynchronisation dieselbe Fehlerannahme machen wie für die byzantinische Verteilung und erhält damit das Problem der **byzantinischen Uhrensynchronisation** [LaMe_84]. Der Konsistenz entspricht die konstant begrenzte Abweichung der lokalen Uhren untereinander (durch Δ_C), der Korrektheit die Begrenzung der Drift (durch Δ_D). Hinsichtlich der Lösbarkeit dieses Problems sind ähnliche Aussagen wie für die byzantinische Verteilung bekannt:

- Fehlerfreie Lösungen existieren dann und nur dann, wenn $t < n/3$ gilt [DoHS_84].
- Setzt man die Existenz eines sicheren Authentikationsmechanismus voraus, so ist jedes $t < n$ tolerierbar [HSSD_84].[20]

In einem *vollständig asynchronen* verteilten System können Prozessoren beliebig langsam arbeiten und kann die Übertragung einer Nachricht beliebig lange dauern. Es wird nur vorausgesetzt, daß ein korrekter Prozessor, solange er noch nicht terminiert ist, stets irgendwann einen Schritt macht und daß jede abgesendete Nachricht irgendwann am Ziel ankommt.

Unter diesen Voraussetzungen ist byzantinische Verteilung deterministisch überhaupt nicht erreichbar, selbst dann nicht, wenn man nur von höchstens einem ($t = 1$) Anhalteausfall ausgeht [FiLP_85].

Mit einem beliebig kleinen Fehler können probabilistische Protokolle das **asynchrone byzantinische Verteilungsproblem** lösen [BenO_83, Brac_87] (auch [Rabi_83]).

Der unvermeidliche Fehler wird meist als **Terminierungsfehler** formuliert: Für jede Rundenzahl R (wobei Runde hier für jeden Prozessor lokal definiert wird) ist die Wahrscheinlichkeit $\varepsilon(R)$, daß in der R-ten Runde noch nicht alle korrekten Prozessoren terminiert sind, positiv. Strebt R gegen Unendlich, so strebt $\varepsilon(R)$ aber gegen 0. Terminieren alle Prozessoren, so ist die gefundene Lösung korrekt und konsistent.

Asynchrone byzantinische Verteilung ist in diesem Sinne dann und nur dann möglich, wenn $t < n/3$ gilt [Brac_87, BrTo_85]. Dies gilt auch unter der Annahme

---

[20] Soll die Drift der synchronisierten Uhren ebenso klein sein wie die der unsynchronisierten lokalen Uhren, so ist nur $t < n/2$ tolerierbar [SrTo1_87].

eines sicheren Authentikationsmechanismus [BrTo_85][21]. Unter der Annahme, daß nur Anhalteausfälle auftreten, ist im besten Fall jedes $t < n/2$ tolerierbar [BrTo_85].

Legt man für ein solches Protokoll a priori eine maximale Rundenanzahl R fest und bricht ein Prozessor für den Fall, daß er nach R Runden immer noch nicht terminiert ist, das Protokoll lokal ab, so erhält man ein Protokoll, das stets terminiert, aber mit Wahrscheinlichkeit $\varepsilon(R) > 0$ ein falsches Ergebnis liefert. Wegen $\lim_{R\to\infty} \varepsilon(R) = 0$ wird dieser Fehler mit hinreichend großem R beliebig klein.

1.4.2 Sichere und nichtabhörbare Verbindungen

Die stärkste Annahme des byzantinischen Fehlermodells ist die **Zusammenhangsannahme**, d. h. die der Existenz sicherer Verbindungen zwischen allen korrekten Prozessoren.

Diese Annahme wird zwar durch übliche Kommunikationsprotokolle unterstützt [Tane_88], kann letztendlich aber nur durch physikalische Maßnahmen gerechtfertigt werden und bedeutet praktisch, daß Verbindungsfehler im byzantinischen Fehlermodell *nicht* betrachtet werden. Wird eine Verbindung untolerierbar fehlerhaft, so werden damit zugleich die bislang verbundenen Prozessoren zu Störern.

Die Zusammenhangsannahme ist zur Lösung des byzantinischen Verteilungsproblems jedoch notwendig: Sind die korrekten Prozessoren in zwei Mengen partitioniert, die nicht mehr miteinander kommunizieren können, so kann ein korrekter Transmitter in der einen Partition die korrekten Prozessoren in der anderen nicht mehr erreichen.

Die Implementierung sicherer bzw. sicherer nichtabhörbarer Verbindungen kann zu einem guten Teil auf das Vorbereitungsprotokoll verlagert werden:

Man betrachte zwei beliebige Prozessoren P, Q. Durch Authentikationscodes [GiMS_74] (vgl. Abschnitt 1.1.2.2.1) kann die Authentizität der zwischen P und Q ausgetauschten Nachrichten mit exponentiell kleiner Fehlerwahrscheinlichkeit verifiziert werden (sichergestellt natürlich nicht). Durch eine Vernam-Chiffre (*one-time-pad*) können die ausgetauschten Nachrichten perfekt geheimgehalten werden (ohne jede Fehlerwahrscheinlichkeit) [Shan_49]. Physikalisch zu realisieren bleibt damit nur die Integrität der Verbindung.

Damit P und Q Authentikationscodes und Vernam-Chiffren einsetzen können, müssen sie für jede später auszutauschende Nachricht im Vorbereitungsprotokoll sicher und nichtabhörbar einen entsprechend langen Schlüssel

---

21 Die Anzahl der im byzantinischen Fehlfunktionsmodell tolerierbaren Störer kann daher hier, im Gegensatz zur üblichen, synchronen byzantinischen Verteilung, durch die in Abschnitt 3.1 eingeführten Pseudosignaturen nicht erhöht werden.

wählen und austauschen.

Für kleine Werte von t kann die Forderung nach sicheren und sogar die nach sicheren nichtabhörbaren Verbindungen zwischen jeweils *allen* Prozessoren abgeschwächt werden:

Betrachtet wird der ungerichtete Graph, dessen Knoten den Prozessoren und dessen Kanten den sicheren (nichtabhörbaren) Verbindungen entsprechen.

Sollen die *sicheren* Verbindungen fehlerfrei und ohne vorherigen Schlüsselaustausch realisiert werden, so muß es zwischen je zwei korrekten Prozessoren wenigstens $2t+1$ knotendisjunkte Wege in diesem Graphen geben [Dole_82]. Zum Senden von P an Q sendet P seine Nachricht über alle $2t+1$ Wege an Q, wenigstens $t+1$ der Kopien kommen an, und Q kann aus allen bis zu $2t+1$ erhaltenen Nachrichten die richtige als die mehrheitlich empfangene auswählen [LaSP_82]. Unter derselben Voraussetzung können auch sichere *nichtabhörbare* Verbindungen realisiert werden (*secure message passing*) [DDWY_90].

Haben im Vorbereitungsprotokoll P und Q Schlüssel eines Authentikationscodes ausgetauscht und ist eine exponentiell kleine Fehlerwahrscheinlichkeit akzeptabel, so genügt es, wenn es zwischen je zwei korrekten Prozessoren wenigstens $t+1$ knotendisjunkte Wege gibt. Q kann die korrekte Nachricht von P anhand des Authentikationscodes erkennen.

Die Forderung nach $2t+1$ (bzw. $t+1$) knotendisjunkten Wegen impliziert, daß jeder Knoten mit wenigstens $2t+1$ (bzw. $t+1$) anderen direkt verbunden ist. Für $t \geq n/2-1$ (bzw. $t \geq n-2$) impliziert dies wieder einen vollvermaschten Graphen.

1.4.3 Vorbereitungsprotokoll mit zuverlässigem Verteilkanal

Der im Vorbereitungsprotokoll benötigte zuverlässige Verteilkanal muß, wie die sicheren Verbindungen zwischen den Prozessoren, letztendlich physikalisch realisiert werden:

Die römischen Zenturionen im historischen Beispiel hätten sich in Rom in einem Zelt treffen und ihre zu verteilenden Nachrichten laut gegenseitig zurufen können. Echte Prozessoren können im — allerdings ziemlich unrealistischen — Idealfall zur Initialisierung ein physikalisches Verteilnetz verwenden, etwa ein Funk- oder Busnetz, das dann für die eigentliche Protokollausführung nicht mehr zur Verfügung steht.

Stellt man sich die Prozessoren realistischer als Rechner vor, die man nur schwer mit einem zuverlässigen physikalischen Verteilnetz verbinden kann, so können die Betreiber dieser Rechner das Vorbereitungsprotokoll auch mit anderen Rechnern in einer sichereren Umgebung ausführen und das Ergebnis ihren jeweiligen „echten“ Rechnern zusammen mit dem eigentlichen Verteilungsprotokoll einfach mitteilen. Besonders einfach ist dies dann, wenn alle Rechner denselben Betreiber (aber evtl. verschiedene, einander mißtrauende

Benutzer) haben. (Üblicherweise stellt man sich vor, daß so die Testschlüssel eines digitalen Signatursystems für authentisierte Protokolle verteilt werden.)

Akzeptiert man für das Vorbereitungsprotokoll ein schwächeres Fehlermodell als später, so kann man natürlich auch übliche Verteilungsprotokolle verwenden: Gibt es anfangs weniger als $n/3$ Störer, so benötigt man dann überhaupt keinen zuverlässigen Verteilkanal. Nimmt man an, daß die Störer am Anfang noch keine digitalen Signaturen brechen können, so kann man ein authentisiertes Verteilungsprotokoll verwenden. Der zuverlässige Verteilkanal wird dann nur dazu gebraucht, die Testschlüssel des Signatursystems zuverlässig zu verteilen (zur Vorbereitung der folgenden Verteilungsprotokolle muß sehr viel mehr Information zuverlässig verteilt werden).

Die Annahme, daß die Störer anfangs in geringerer Zahl oder Mächtigkeit auftreten als später, ist allerdings zumindest für bösartige Störer wenig plausibel.

Eine sinnvollere Möglichkeit ist, ein authentisiertes Verteilungsprotokoll mit den in Abschnitt 1.1.2.3 erwähnten Fail-stop-Signaturen zu verwenden:

Wird das authentisierte Verteilungsprotokoll untolerierbar gestört, so bedeutet dies, daß die Störer Signaturen korrekter Prozessoren gefälscht haben. Dies kann aber in einem Fail-stop-Signatursystem (mit exponentiell kleiner Fehlerwahrscheinlichkeit) festgestellt werden. Kann also am Ende des Vorbereitungsprotokolls kein Prozessor beweisen, daß seine Signaturen gefälscht wurden, so ist sicher (bis auf einen exponentiell kleinen Fehler), daß das Vorbereitungsprotokoll korrekt ausgeführt wurde [PfWa1_91, WaPf_89].

Man hat damit zwar keine Erfolgsgarantie, zumindest aber einen *Erfolgstest*. Aus Aufwandsgründen ist ein praktischer Einsatz dieser Möglichkeit zur Zeit allerdings nur schwer vorstellbar.

1.4.4 Fehlerwahrscheinlichkeit

Zum Abschluß sei noch bemerkt, daß eine beliebig kleine Fehlerwahrscheinlichkeit praktisch keine Einschränkung bedeutet:

Implementiert man ein Protokoll, so entsprechen den hier betrachteten Prozessoren und Verbindungen reale Rechner und Leitungen, die wie jedes physikalische Gerät mit einer gewissen Wahrscheinlichkeit ausfallen. Eine Fehlerwahrscheinlichkeit = 0 anzustreben, ist also sinnlos.

Erlaubt die Anwendung eines Protokolls nur eine bestimmte Fehlerwahrscheinlichkeit ε, so muß diese zwischen dem Protokoll und dem physikalischen System aufgeteilt werden. Im folgenden wird in Beispielen willkürlich 2^{-100} ($\approx 10^{-30}$) als hinreichend kleine Fehlerwahrscheinlichkeit eines Protokolls betrachtet.

1.5 Überblick über die Arbeit

Im byzantinischen Fehlermodell (Abschnitt 1.2) konnten bislang nur $t < n/3$ Störer toleriert werden (Abschnitt 1.3). Mittels digitaler Signaturen konnten durch authentisierte Verteilungsprotokolle zwar $t < n$ Störer toleriert werden, jedoch nur um den Preis, das byzantinische zum kryptographischen Fehlfunktionsmodell abzuschwächen (Abschnitt 1.1.2.2).

In den folgenden Kapiteln werden erstmals Verteilungsprotokolle vorgestellt, die im byzantinischen Fehlermodell ein $t \geq n/3$ tolerieren.

Die binären Verteilungsprotokolle aus Kapitel 2 tolerieren jedes $t < n/2$, setzen also lediglich eine korrekte Mehrheit voraus.

Grundlage ist ein Protokoll zur überprüften Geheimnisaufteilung (*verifiable secret sharing*, Abschnitt 2.1.1), das ebenfalls nur eine korrekte Mehrheit voraussetzt und das zum Zusammensetzen eines Geheimnisses *keinen* zuverlässigen Verteilkanal benötigt. Ein solches Verfahren war bislang ebenfalls unbekannt; ein Protokoll aus [RaBe_89] (Abschnitt 2.1.2) konnte aber so verbessert und modifiziert werden, daß es diesen Anforderungen genügt (Abschnitte 2.1.3 und 2.1.4).

Das byzantinische Verteilungsprotokoll aus Abschnitt 2.2.3 hat eine Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ für ein beliebig gewähltes σ. Es benötigt im schlimmsten Fall $O(\sigma)$ Runden, im Erwartungswert aber weniger als 19 Runden. Hinsichtlich seiner Rundenanzahl ist es daher ebenso effizient wie die effizientesten probabilistischen Protokolle für $t < n/3$ [FeMi_88].

Das Verteilungsprotokoll aus Kapitel 3 toleriert jedes $t < n$ und bietet damit das Maximum an Fehlertoleranz im byzantinischen Fehlermodell.

Grundlage ist ein im byzantinischen Fehlermodell sicherer Authentikationsmechanismus, *Pseudosignaturen* genannt, der in vielen Protokollen statt der üblichen, nur kryptographisch sicheren digitalen Signaturen verwendet werden kann (Abschnitt 3.1). Ein solcher Authentikationsmechanismus war bislang unbekannt. Pseudosignaturen sind eine Verallgemeinerung eines in [ChRo_91] beschriebenen Authentikationsmechanismus.

Das in Abschnitt 3.2 beschriebene Verteilungsprotokoll basiert auf einem bekannten authentisierten Verteilungsprotokoll [DoSt_83], in das Pseudosignaturen statt echter digitaler Signaturen passend eingesetzt werden. Das Protokoll hat eine Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ und benötigt, wie das Original, deterministisch $O(t)$ Runden. Für $t < n/2$ ist das Verteilungsprotokoll aus Abschnitt 2.2.4 rundenärmer.

Die Verteilungsprotokolle aus Abschnitt 2.2 für korrekte Mehrheiten können jedoch nur binäre Nachrichten verteilen, d. h. um z-bit-Nachrichten zu verteilen, müßte man das jeweilige Verteilungsprotokoll z mal ausführen.

In Kapitel 4 wird daher mit Hilfe der Pseudosignaturen aus Abschnitt 3.1 und

einer für normale digitale Signaturen gedachten Konstruktion von TOUEG die Verteilung langer Nachrichten effizienter auf die Verteilung binärer Nachrichten reduziert. Das Ergebnis toleriert jedes $t < n/2$ und benötigt deterministisch 4 Runden mehr als das ursprüngliche binäre Verteilungsprotokoll, im Erwartungswert also weniger als 23 Runden.

Alle hier beschriebenen Verteilungsprotokolle setzen ein Vorbereitungsprotokoll voraus, in dem die Prozessoren, im Gegensatz zum eigentlichen Verteilungsprotokoll, gewisse Informationen zuverlässig an alle verteilen können. (Dies entspricht etwa der Schlüsselaustauschphase üblicher digitaler Signaturen.)

Die Vorbereitung von u Ausführungen eines Verteilungsprotokolls erfordert zunächst die u-malige Ausführung des zugehörigen Vorbereitungsprotokolls. In Abschnitt 3.2.3 bzw. Abschnitt 4.3 wird daher gezeigt, wie das Verteilungsprotokoll aus Abschnitt 3.2.1 bzw. Abschnitt 4.2 u mal ausgeführt werden kann, so daß u nur logarithmisch in den Aufwand des Vorbereitungsprotokolls mit zuverlässigem Verteilkanal eingeht.

In Kapitel 5 wird als wichtige theoretische Anwendung der Verteilungsprotokolle das Problem der sicheren Mehrparteienberechnungen beschrieben.

2 Byzantinische Verteilung mit korrekter Mehrheit in erwartet konstanter Rundenanzahl

Im folgenden wird ein Protokoll zur (binären) byzantinischen Verteilung beschrieben, das im byzantinischen Fehlermodell jedes $t < n/2$ in erwartet weniger als 19 Runden mit exponentiell kleiner Fehlerwahrscheinlichkeit toleriert.

Das Protokoll ist das erste, das dieses Maß an Fehlertoleranz erreicht (bislang konnte im byzantinischen Fehlermodell nur jedes $t < n/3$ toleriert werden).

Wesentlichstes Hilfsmittel ist ein Protokoll zur *überprüften Geheimnisaufteilung* (*verifiable secret sharing*) für $t < n/2$, das in Abschnitt 2.1 ausführlich beschrieben wird. Das eigentliche Verteilungsprotokoll wird dann in Abschnitt 2.2 beschrieben.[22]

Notation: Zu einer Primzahlpotenz p sei GF(p) der endliche Körper mit p Elementen.[23] Zu einem endlichen Körper F sei Π_t die Menge aller Polynome $f \in F[X]$ vom Grade $\leq t$, d. h.

$$\Pi_t := \{ f \mid f(X) = \gamma_0 + \gamma_1 X + \ldots + \gamma_t X^t, \gamma_0, \gamma_1, \ldots, \gamma_t \in F \},$$

und für jedes $s \in F$ sei $\Pi_t(s)$ die Menge aller Polynome $f \in \Pi_t$ mit $f(0) = s$, d. h.

$$\Pi_t(s) := \{ f \mid f(X) = s + \gamma_1 X + \ldots + \gamma_t X^t , \gamma_1, \ldots, \gamma_t \in F \}.$$

Sei $\sigma \in \mathbb{N}$ ein **Sicherheitsparameter**, so daß eine Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ vernachlässigbar ist.

Wählt ein Prozessor **zufällig** ein Element x aus einer endlichen Menge M, so bedeutet dies, daß er x gemäß einer Gleichverteilung und unabhängig von allen anderen Zufallsauswahlen aus M wählt.

2.1 Überprüfte Geheimnisaufteilung

Das in Abschnitt 2.2 verwendete Protokoll zur überprüften Geheimnisaufteilung basiert auf einem Protokoll zur überprüften Geheimnisaufteilung von TAL RABIN [RaBe_89, Rabi_88].

RABINs Protokoll setzt einen zuverlässigen Verteilkanal voraus und kann

---

22 Teile des Inhalts dieses Kapitels (außer Abschnitt 2.1.2.5 und Abschnitt 2.2.3) sind auch in [BaPW_91] beschrieben.

23 Eine Erläuterung dieser und der folgenden algebraischen Begriffe findet man z. B. in [LiNi_86].

daher nicht direkt in einem Verteilungsprotokoll verwendet werden. Es kann aber so modifiziert werden, daß der zuverlässige Verteilkanal nur noch in einem Teil des Protokolls benötigt wird, der dann im Verteilungsprotokoll aus Abschnitt 2.2.1 während des Vorbereitungsprotokolls ausgeführt werden kann.

Abschnitt 2.1.1 beschreibt informell, was ein Protokoll zur überprüften Geheimnisaufteilung erreicht und was zum Verständnis des folgenden unverzichtbar ist. Abschnitt 2.1.2 beschreibt RABINs Protokoll und Abschnitt 2.1.3 die für Abschnitt 2.2 benötigte Modifikation.

2.1.1 Überblick über überprüfte Geheimnisaufteilung

Mit einem Protokoll zur **Geheimnisaufteilung** (*secret sharing*) mit **Schwellwert** t kann ein Prozessor, der **Geheimniskrämer** G (*dealer*), ein **Geheimnis** s so auf alle Prozessoren aufteilen, daß ohne Hilfe anderer Prozessoren

- *keine* Menge von höchstens t Prozessoren irgendwelche Information (im SHANNONschen Sinne) über das Geheimnis s erhält,
- aber *jede* Menge von mindestens $t+1$ Prozessoren das Geheimnis s eindeutig und effizient bestimmen kann [Blak_79, Sham_79].

Ein Protokoll zur Geheimnisaufteilung besteht aus zwei Phasen, eine zum **Aufteilen** eines Geheimnisses in n **Anteile** (ein Anteil je Prozessor) und eine zum **Zusammensetzen** eines Geheimnisses aus wenigstens $t+1$ Anteilen.

Geheimnisaufteilung ist nur für $t < n/2$ sinnvoll: Die höchstens t Störer sollen keine Information über ein Geheimnis s erhalten, die mindestens $n-t$ korrekten Prozessoren hingegen sollen s zusammensetzen können. Folglich muß $t < n-t$ gelten, d. h. $t < n/2$.

Protokolle zur Geheimnisaufteilung sind nicht notwendigerweise fehlertolerant [CGMA_85]: Ein fehlerhafter Geheimniskrämer kann unbemerkt Anteile verteilen, die nicht zusammenpassen und daher kein eindeutiges Geheimnis definieren. Störer können ihre Anteile unbemerkt abändern, so daß ein falsches Geheimnis zusammengesetzt wird.

Protokolle zur **überprüften Geheimnisaufteilung** (*verifiable secret sharing*) lösen diese Probleme [CGMA_85]: Am Ende des Aufteilens eines Geheimnisses bewerten die korrekten Prozessoren übereinstimmend das Ergebnis mit „akzeptiert“ oder „abgelehnt“. Ein korrekter Prozessor akzeptiert nur dann, wenn genau ein Geheimnis existiert, das unabhängig vom Verhalten der Störer später zusammengesetzt werden kann.

Wie byzantinische Verteilung, so kann auch überprüfte Geheimnisaufteilung ohne Fehlerwahrscheinlichkeit im byzantinischen Fehlermodell nur für $t < n/3$ realisiert werden [BeGW_88]. Dies gilt selbst dann, wenn ein zuverlässiger Verteilkanal verfügbar ist [ChCD_88, DDWY_90].

RABINs Protokoll ist das erste, das im byzantinischen Fehlermodell jedes $t < n/2$ mit einer beliebig kleinen Fehlerwahrscheinlichkeit toleriert [RaBe_89].

Sein Aufwand ist polynomial in n und einem Sicherheitsparameter τ. Die Aufteilung eines Geheimnisses benötigt eine in n lineare, die Zusammensetzung eine konstante Rundenanzahl.[24] Die Fehlerwahrscheinlichkeit des Protokolls ist $\leq 2^{-\tau}$, also exponentiell klein. Das Protokoll wird in Abschnitt 2.1.2 genau beschrieben.

RABINs Protokoll benötigt einen zuverlässigen Verteilkanal, sowohl zum Aufteilen als auch zum Zusammensetzen von Geheimnissen. In Abschnitt 2.1.3 wird ein modifiziertes Protokoll dargestellt, das zum Zusammensetzen *keinen* zuverlässigen Verteilkanal benötigt. Davon abgesehen leistet es dasselbe wie RABINs Protokoll und verursacht denselben Aufwand.

Kurzfassung für eilige Leser: Wem das Lesen des Rests von Abschnitt 2.1 zu mühsam erscheint, möge folgende Fakten über das modifizierte Protokoll zur überprüften Geheimnisaufteilung glauben und kann dann ab der Beschreibung der byzantinischen Verteilungsprotokolle, d. h. ab Abschnitt 2.2, weiterlesen:

Für das modifizierte Protokoll wird ein großer endlicher Körper F gewählt, insbesondere einer mit $|F| > 2^{\tau}$ (genauer vgl. Abschnitt 2.1.3.3). Nachdem ein Geheimnis $s \in F$ erfolgreich aufgeteilt wurde, hat jeder korrekte Prozessor P_i vom Geheimniskrämer G einen überprüften Anteil $\beta_i \in F$ an s erhalten. Zusammen mit β_i hat P_i gewisse *Authentikationsnachweise* erhalten, die fast sicher jeden anderen korrekten Prozessor von der Authentizität seines Anteils β_i überzeugen können. Erhält ein Prozessor Anteile und Authentikationsnachweise von anderen Prozessoren, so kann er fast sicher entscheiden, welche der erhaltenen Anteile authentisch sind. Erhält ein Prozessor wenigstens $t+1$ authentische Anteile, so kann er das korrekte Geheimnis fast sicher zusammensetzen. Fast sicher meint dabei jeweils mit Wahrscheinlichkeit $\geq 1–2^{-\tau}$.

Aus Effizienzgründen wird in Abschnitt 2.2 neben dem modifizierten Protokoll zur überprüften Geheimnisaufteilung (Protokoll 2.6) ein Protokoll zur schwachen Geheimnisfestlegung (Protokoll 2.5) verwendet. Um die Verteilungsprotokolle in Abschnitt 2.2 verstehen zu können, kann man in ihnen Protokoll 2.5 überall durch Protokoll 2.6 ersetzen.

---

24 Das Protokoll wie in [RaBe_89, Rabi_88] beschrieben benötigt auch zum Aufteilen nur eine konstante Rundenanzahl, enthält aber leider einen kleinen Entwurfsfehler (vgl. Anmerkung nach Protokoll 2.4). In Abschnitt 2.1.2 wird das verbesserte Protokoll ausführlich beschrieben und bewiesen. Mit „RABINs Protokoll“ ist im folgenden stets dieses verbesserte Protokoll gemeint.

2.1.2 RABINs Protokoll zur überprüften Geheimnisaufteilung

Fast alle bekannten Geheimnisaufteilungsprotokolle basieren auf dem sehr einfachen Geheimnisaufteilungsprotokoll von SHAMIR (Protokoll 2.1, Abschnitt 2.1.2.1). Dieses wird in [RaBe_89] so erweitert, daß ein Protokoll zur überprüften Geheimnisaufteilung entsteht:

Der Geheimniskrämer G verteilt s zunächst wie in SHAMIRs Geheimnisaufteilungsprotokoll. Jeder Prozessor P_i legt sich sodann auf den von G erhaltenen Anteil β_i fest, ohne jedoch irgendwelche Information über β_i preiszugeben. Diese Festlegung erfolgt durch eine Variante der Geheimnisaufteilung, genannt Geheimnisfestlegung (*weak secret sharing*). Die Korrektheit beider Ebenen der Geheimnisaufteilung wird so überprüft, daß die Störer hierdurch keine Information erhalten (*Zero-knowledge*-Beweis).

Zum Zusammensetzen des Geheimnisses werden anhand der Geheimnisfestlegungen authentische Anteile identifiziert, darunter wenigstens die $n-t > t$ Anteile der korrekten Prozessoren, und aus diesen s zusammengesetzt.

Die exakte Konstruktion wird im folgenden schrittweise beschrieben, beginnend bei den benötigten Grundprotokollen (also *bottom-up*).

Die dargestellte Konstruktion entspricht im wesentlichen der ursprünglichen von RABIN. In Protokoll 2.4 mußte ein Entwurfsfehler verbessert werden (vgl. Anmerkung nach Protokoll 2.4). Als Folge davon benötigt RABINs Protokoll hier $O(t)$ Runden (statt $O(1)$ wie in [RaBe_89]). In Abschnitt 2.1.2.5 wird aber beschrieben, wie für u sequentielle Ausführungen von RABINs Protokoll die Rundenanzahl auf $O(u+n^2)$ gesenkt werden kann; für $u = n^c$, $c \geq 2$, und hinreichend große n sind dies im Mittel je Ausführung wieder konstant viele.

Die Originalarbeiten [RaBe_89, Rabi_88] enthalten leider keine vollständigen Theoreme und Beweise und so gut wie keine Fehler- oder Aufwandsbetrachtungen. Deshalb, und da das Protokoll wegen des o.g. Entwurfsfehlers ohnehin geändert werden mußte, werden diese formalen Betrachtungen hier nachgeholt.

RABINs Protokoll setzt zum Aufteilen und Zusammensetzen eines Geheimnisses einen zuverlässigen Verteilkanal voraus. **Veröffentlicht** ein Prozessor P eine Nachricht, so heißt dies, daß P die Nachricht über den zuverlässigen Verteilkanal sendet. (Zum Aufteilen werden zusätzlich sichere nichtabhörbare Verbindungen benötigt, zum Zusammensetzen genügen sichere Verbindungen.)

Dank des zuverlässigen Verteilkanals können in den folgenden Protokollen Störer (und fast sicher nur Störer) öffentlich **disqualifiziert** werden, d.h. so, daß alle korrekten Prozessoren sich stets einig sind, welche Prozessoren disqualifiziert wurden und welche nicht. Disqualifizierte Prozessoren werden von den korrekten ignoriert, d.h. sie nehmen nicht mehr am Protokoll teil. Implizit wird in den folgenden Protokollen angenommen, daß jeder Prozessor

disqualifiziert wird, der sich für *alle* offensichtlich nicht an sein Protokoll hält, also z. B. eine erwartete Nachricht nicht oder falsch veröffentlicht. (Muß ein Prozessor P_i über eine sichere Verbindung an einen Prozessor P_j eine Nachricht senden und tut dies nicht oder falsch, so kann dies nur P_j feststellen. Beschwert sich P_j, so ist im allgemeinen nicht klar, ob P_i fehlerhaft ist oder P_j. P_i darf daher *nicht* öffentlich disqualifiziert werden. Der enttäuschte Empfänger kann allerdings P_i *lokal* disqualifizieren.)

2.1.2.1 SHAMIRs Geheimnisaufteilungsprotokoll

RABINs Protokoll basiert auf einem einfachen Protokoll zur Geheimnisaufteilung (*secret sharing*) [Blak_79, Sham_79]:

Sei F ein endlicher Körper mit $|F| \geq n+1$ und seien $\alpha_1, \ldots, \alpha_n$ beliebige, aber paarweise verschiedene Elemente aus $F \setminus \{0\}$.

Protokoll 2.1: SHAMIRs Geheimnisaufteilungsprotokoll

Jeder Prozessor P_i hat als lokale Eingabe seinen Index i,
der Geheimniskrämer G zusätzlich sein Geheimnis $s \in F$.

Phase 1: Aufteilen des Geheimnisses

G wählt zufällig ein Polynom f aus $\Pi_t(s)$, d. h. er wählt t Koeffizienten $\gamma_1, \ldots, \gamma_t$ zufällig aus F und setzt

$$f(X) := s + \gamma_1 X + \ldots + \gamma_t X^t.$$

G sendet an jeden Prozessor P_i, $i = 1, \ldots, n$, den **Anteil** β_i mit

$$\beta_i := f(\alpha_i).$$

Phase 2: Zusammensetzen eines Geheimnisses

Jeder Prozessor, der wenigstens $t+1$ Anteile

$$(\alpha_{i_1}, \beta_{i_1}), \ldots, (\alpha_{i_{t+1}}, \beta_{i_{t+1}})$$

kennt, interpoliert f an der Stelle 0, etwa nach LAGRANGE durch

$$f(X) = \sum_{j=1}^{t+1} \frac{(X - \alpha_{i_1}) \ \ldots \ (X - \alpha_{i_{j-1}}) \ (X - \alpha_{i_{j+1}}) \ \ldots \ (X - \alpha_{i_{t+1}})}{(\alpha_{i_j} - \alpha_{i_1}) \ \ldots \ (\alpha_{i_j} - \alpha_{i_{j-1}}) \ (\alpha_{i_j} - \alpha_{i_{j+1}}) \ \ldots \ (\alpha_{i_j} - \alpha_{i_{t+1}})} \ \beta_{i_j},$$

und erhält $s = f(0)$.

Beispiel: Man betrachte $n = 5$ Prozessoren und bis zu $t = 2$ Störer. Als Körper wird $F = \mathrm{GF}(7)$ verwendet (d.h. es wird modulo 7 gerechnet), und die benötigten 5 Körperelemente seien

$$(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5) := (1, 2, 3, 4, 5).$$

Der Geheimniskrämer hat als lokale Eingabe das Geheimnis $s = 4$. In Phase 1 wählt er zufällig ein Polynom vom Grade $\leq t+1 = 3$, etwa

$$\gamma_1 := 3,\ \gamma_2 := 1 \text{ und damit } f(X) = 4 + 3X + X^2.$$

Die 5 Prozessoren (einschließlich des Geheimniskrämers) erhalten die Anteile

$$(\beta_1, \beta_2, \beta_3, \beta_4, \beta_5) = (1, 0, 1, 4, 2).$$

In Phase 2 werden die $t+1 = 3$ Anteile (α_1, β_1), (α_2, β_2), (α_3, β_3) zum Zusammensetzen verwendet. Es ist dann

$$f(0) = \frac{(0-2)(0-3)}{(1-2)(1-3)}\,1 + \frac{(0-1)(0-3)}{(2-1)(2-3)}\,0 + \frac{(0-1)(0-2)}{(3-1)(3-2)}\,1 = 3 + 0 + 1 = 4 = s. \qquad \blacktriangleleft$$

Lemma 2.1: (SHAMIRs Geheimnisaufteilungsprotokoll)

a) Ist G korrekt, so erhalten die maximal t Störer vor Phase 2 keine Information über s.

b) Ist G korrekt und werden in Phase 2 solche $t+1$ Anteile verwendet, die G in Phase 1 verteilt hat, so wird das korrekte Geheimnis s zusammengesetzt.

c) Der Rechenaufwand des Protokolls ist polynomial in n und $\log(|F|)$. ♦

Die Aussagen von Lemma 2.1 gelten jeweils *ohne* Fehlerwahrscheinlichkeiten.

Beweis: (nach [Sham_79])

a) Sei M eine beliebige Menge von t Anteilen. Nach LAGRANGE bestimmen $t+1$ beliebige Anteile genau ein Polynom aus Π_t [LiNi_86: Th. 1.71]. Für jedes $s' \in F$ gibt es daher genau ein Polynom $f_{s'} \in \Pi_t$, das zu $f_{s'}(0) = s'$ und den t Anteilen in M paßt. Folglich enthält M keine Information über s. Dies gilt erst recht für jede Teilmenge von M. Folglich enthält auch die Menge der höchstens t den Störern bekannten Anteile keine Information über s.

b) Nach LAGRANGE bestimmen jeweils $t+1$ Anteile das ursprüngliche Polynom f und damit auch $s = f(0)$ [LiNi_86: Th. 1.71].

c) Der Aufwand für arithmetische Operationen und Polynominterpolation in F ist polynomial in n und $\log(|F|)$. ❒

2.1.2.2 RABINs Authentikationsnachweisprotokoll

Die Basis von RABINs Protokoll ist ein **Authentikationsnachweisprotokoll** (*information checking*).

Betrachtet werden drei Prozessoren (Bild 2-1), der Geheimniskrämer G, ein Zwischenprozessor Z und ein Empfängerprozessor E, sowie ein Geheimnis s. Z erhält s von G. Solange bis s von Z an E übergeben wird, soll s perfekt geheim

sein. E soll nur das authentische s von Z akzeptieren, und Z soll vorhersehen können, ob E das Geheimnis akzeptieren wird oder nicht.

Mit üblichen digitalen Signaturen (vgl. Abschnitt 1.1.2.2.2) könnte man dieses Problem sehr einfach lösen: G würde s signieren, und sowohl Z als auch E könnten die Authentizität von s anhand der Signatur überprüfen. Ein Authentikationsnachweis löst das Problem, ohne kryptographische Annahmen machen zu müssen [RaBe_89: WSS].

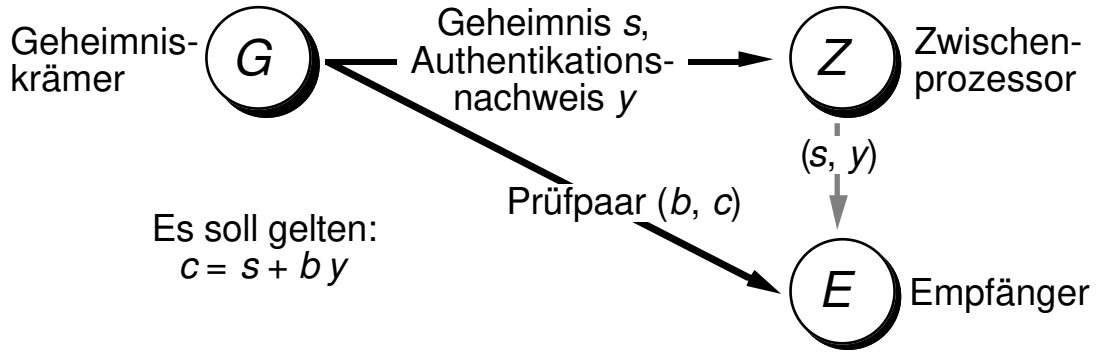


Bild 2-1 RABINs Authentikationsnachweisprotokoll (Protokoll 2.2). Die schwarzen Pfeile beziehen sich auf Phase 1, Schritt [1], der graue Pfeil auf Phase 2. Das Überprüfen von (b, c) in Phase 1, Schritte [2] und [3] ist nicht dargestellt.

Die Fehlerwahrscheinlichkeit des Authentikationsnachweisprotokolls (und damit auch aller nachfolgenden Protokolle) wird durch einen Parameter k bestimmt. Um einen Fehler $\leq 2^{-k}$ zu erreichen, muß gelten $|F| \geq k\,2^{k+1}$, und um später die Geheimnisaufteilung zu ermöglichen, muß gelten $|F| \geq n+1$.

Im folgenden wird stets angenommen, daß $k \geq \log_2(n)$ ist. $|F| \geq k\,2^{k+1}$ impliziert dann (für $n \geq 1$) $|F| \geq n+1$. Dies schadet nichts, denn ist $k < \log_2(n)$, so ist die Fehlerwahrscheinlichkeit $2^{-k} > n^{-1}$, also nicht exponentiell klein.

Zur Vereinfachung wird angenommen, daß zu gegebenen n und k der endliche Körper

$$F^{(k)} := \mathrm{GF}(2^{u_k}) \text{ mit } u_k := \lceil k + \log_2(k) + 1 \rceil.$$

verwendet wird.

Um in $F^{(k)}$ rechnen zu können, muß ein irreduzibles Polynom vom Grade u_k über GF(2) bestimmt werden. Ein in u_k (und damit in n und k) erwartet polynomialer probabilistischer Algorithmus hierfür ist in [Rabi_80] beschrieben. Der Aufwand zur Bestimmung von $F^{(k)}$ wird in den folgenden Protokollen nicht explizit betrachtet, da $F^{(k)}$ ja insgesamt nur einmal bestimmt werden muß.

Die Codierung der Elemente von $F^{(k)}$ erfordert u_k bit = $O(k)$ bit, denn man kann $F^{(k)}$ mit der Menge aller Vektoren aus $\{0,1\}^{u_k}$ identifizieren. Die Arithmetik in $F^{(k)}$ ist polynomial in u_k, d. h. in n und k.

Damit $F^{(k)}$ zur Geheimnisaufteilung wie in Protokoll 2.1 verwendet werden kann, seien für jedes k paarweise verschiedene Elemente

$$\alpha_1, \ldots, \alpha_n \in F^{(k)} \setminus \{0\}$$

uniform festgelegt.

Identifiziert man $F^{(k)}$ mit $\{0,1\}^{u_k}$, so kann man als α_i die Binärdarstellung von i mit u_k Ziffern (mit führenden Nullen) verwenden.

Beispiel: Für $k = 80$ erhält man

$$u_{80} = \lceil 87{,}321\ldots \rceil = 88,$$

d. h. $F^{(80)} = \mathrm{GF}(2^{88})$. Um in F rechnen zu können, benötigt man ein irreduzibles Polynom vom Grade 88 über GF(2). Ein solches ist z. B. [LiNi_86]

$$g(X) = X^{88} + X^8 + X^5 + X^4 + X^3 + X + 1.$$

Die Körperelemente sind die Polynome vom Grade < 88 über GF(2). Die Addition und Multiplikation von Elementen erfolgt durch die entsprechende Addition und Multiplikation der Polynome über GF(2) modulo $g(X)$. ◣

Alternativ kann man auch jede andere Folge $F^{(k)}$ endlicher Körper verwenden, für die gilt: Für jedes k ist $|F^{(k)}| \geq \max\{k\, 2^{k+1}, n+1\}$, die Arithmetik in $F^{(k)}$ ist polynomial in k, und $F^{(k)}$ läßt sich mit $O(k)$ bit codieren.

Protokoll 2.2: RABINs Authentikationsnachweisprotokoll mit Parameter k

Der Geheimniskrämer G hat als lokale Eingabe das Geheimnis $s \in F^{(k)}$.

Phase 1: Vorbereiten und Überprüfen von Prüfpaaren

[1] Der Geheimniskrämer G wählt zufällig Vektoren

$$\boldsymbol{b} \in (F^{(k)} \setminus \{0\})^{2k} \text{ und } \boldsymbol{y} \in (F^{(k)})^{2k}$$

und bestimmt den Vektor $\boldsymbol{c} \in (F^{(k)})^{2k}$ gemäß

$$c_j := s + b_j y_j,\ j = 1, \ldots, 2k.$$

G sendet $(s, \boldsymbol{y})$ an Prozessor Z und das **Prüfpaar** $(\boldsymbol{b}, \boldsymbol{c})$ an Prozessor E. $\boldsymbol{y}$ wird **Authentikationsnachweis** genannt.

[2] Prozessor Z testet, ob das Prüfpaar für E korrekt gebildet wurde:

[2.1] Z wählt zufällig k Indizes $J \subset \{1, \ldots, 2k\}$ und veröffentlicht J.

[2.2] E veröffentlicht als Antwort alle (j, b_j, c_j) mit $j \in J$.

[2.3] G prüft die Antwort von E.

War sie korrekt, so veröffentlicht G eine Bestätigung.

Ansonsten veröffentlicht G ein korrektes Prüfpaar $(\boldsymbol{b}, \boldsymbol{c})$. E übernimmt dann dieses Prüfpaar statt des in [1] erhaltenen.

[2.4] *Z* prüft die Antwort von *E* und die Reaktion von *G*:

Bestätigt *G* die Antwort von *E*, obwohl $c_j \neq s + b_j y_j$ für wenigstens ein $j \in J$, oder verteilt *G* ein neues Prüfpaar (***b***, ***c***), für das $c_j \neq s + b_j y_j$ für wenigstens ein *j* gilt, so lehnt *Z* das Prüfpaar ab, ansonsten akzeptiert er es.

Z veröffentlicht seine Entscheidung „akzeptiert" oder „abgelehnt".

[3] Hat *Z* das Prüfpaar abgelehnt oder wurde disqualifiziert, so veröffentlicht *G* das Geheimnis *s*. In diesem Fall wird *s* **öffentlich** genannt.

Phase 2: Übergabe von *s* an *E*

Falls *s* nicht bereits öffentlich ist, bildet *Z* das *k*-tupel ***y'*** aller y_j mit $j \notin J$ und sendet das Paar (*s*, ***y'***) an *E*.

E akzeptiert *s*, wenn $c_j = s + b_j y_j$ für wenigstens ein $j \notin J$ gilt, oder wenn *s* öffentlich ist.

Der Hauptunterschied zwischen digitalen Signaturen und Authentikationsnachweisen besteht im Zeitpunkt, zu dem ein zuverlässiger Verteilkanal benötigt wird: Für digitale Signaturen wird der zuverlässige Verteilkanal benötigt, um den Schlüssel *STe* zum Testen von Unterschriften zuverlässig zu verteilen. Die zu signierende Nachricht muß zu diesem Zeitpunkt noch nicht bekannt sein. Für das oben beschriebene Protokoll hingegen wird der zuverlässige Verteilkanal noch benötigt, *nachdem* die zu authentisierende Nachricht *s* bekannt ist.

In dieser Hinsicht werden die in Abschnitt 3.1 beschriebenen Pseudosignaturen echten digitalen Signaturen sehr viel ähnlicher sein.

Das in Phase 1, Schritt [2] verwendete Prinzip ist Grundlage vieler *Zero-knowledge*-Techniken (z. B. in [GoMW_91]) und wird meist anschaulich als *Cut-and-choose*-Technik bezeichnet [Rabi_78].

Die Eigenschaften von Protokoll 2.2 werden in Lemma 2.2 zusammengefaßt. Anschaulich besagt es: Ist höchstens einer der drei Prozessoren fehlerhaft, so arbeitet das Protokoll mit Wahrscheinlichkeit $\geq 1–2^{-k}$ wie gewünscht.

Lemma 2.2: (RABINs Authentikationsnachweisprotokoll)

a) Kein korrekter Prozessor wird disqualifiziert.
b) Sind *G* und *Z* korrekt, so akzeptiert *Z* den Authentikationsnachweis in Phase 1, und *s* bleibt bis Phase 2 im informationstheoretischen Sinne geheim.
c) Sind *Z* und *E* korrekt, so gilt mit Wahrscheinlichkeit $\geq 1–2^{-k}$: Wird *G* in Phase 1 nicht disqualifiziert, so akzeptiert *E* das Geheimnis *s* in Phase 2.
d) Sind *G* und *E* korrekt, so gilt mit Wahrscheinlichkeit $\geq 1–2^{-k}$: Akzeptiert *E*

in Phase 2 ein Geheimnis s, so ist es das authentische Geheimnis.

e) Der Rechenaufwand des Protokolls ist polynomial in k.

Phase 1 benötigt 6 Runden, in denen $O(k^2)$ bit veröffentlicht und über sichere nichtabhörbare Verbindungen gesendet werden

Phase 2 benötigt eine Runde, in der $O(k^2)$ bit über eine sichere Verbindung gesendet werden. ♦

Die Aussagen von Lemma 2.2 entsprechen denen in [RaBe_89: Lemma 1 bis 3], allerdings ohne genaue Fehler- und Aufwandsbetrachtung.

Beweis:

a) Gilt offensichtlich.

b) Z akzeptiert in Schritt [2.4] genau dann, wenn für alle in Schritt [2.4] bekannten gültigen Tripel (b_j, c_j, y_j) gilt $c_j = s + b_j y_j$. Ist dies für ein Antworttripel von E nicht erfüllt, so ersetzt G sie in Schritt [2.3] durch korrekte Tripel. Also akzeptiert Z immer.

E erhält in Phase 1 nur die Paare (b_j, c_j), $j = 1, \ldots, 2k$. Für jedes $s' \in F^{(k)}$ gibt es genau ein y'_j, das $c_j = s' + b_j y'_j$ erfüllt, nämlich $y'_j = (c_j - s')\, b_j^{-1}$. Da alle y'_j gleichwahrscheinlich sind, enthält (b_j, c_j) keine Information über s.

c) Ist s öffentlich, so akzeptiert E es stets. Angenommen, s sei nicht öffentlich. Damit G in Phase 1 nicht disqualifiziert wird, aber E in Phase 2 das Geheimnis s nicht akzeptiert, muß für alle $j \in J$ gelten $c_j = s + b_j y_j$ und für alle $j \notin J$ gelten $c_j \neq s + b_j y_j$. Die Entscheidung, welche Gleichungen gelten und welche nicht, fällt G aber schon in Schritt [1], also bevor Z die Menge J zufällig wählt. Die gesuchte Wahrscheinlichkeit ist daher gerade die, daß G zufällig die richtige Menge J errät, also (wie man leicht nachrechnet)

$$\frac{1}{\binom{2k}{k}} \leq 2^{-k}.$$

d) Akzeptiert E ein öffentliches s, so ist dieses sicher das authentische (denn es stammt von G). Angenommen, s sei nicht öffentlich. Für jedes $j \notin J$ weiß Z nach Phase 1 nur, daß $c_j = s + b_j y_j$ gilt. Damit E ein Geheimnis $s' \neq s$ akzeptiert, muß Z ein y'_j finden, so daß gilt:

$$c_j = s + b_j y_j \;\wedge\; c_j = s' + b_j y'_j$$

$$\Leftrightarrow c_j = s + b_j y_j \;\wedge\; y_j \neq y'_j \;\wedge\; b_j = \frac{s - s'}{y'_j - y_j}$$

Für jedes Paar (s', y'_j) mit $s \neq s'$ und $y_j \neq y'_j$ gibt es also genau ein Prüfpaar (b_j, c_j), das zu (s, y_j) paßt, und mit dem E auch (s', y'_j) akzeptieren würde. (Zu (s', y'_j) mit $s \neq s'$ und $y_j = y'_j$ gibt es kein Prüfpaar.) Insgesamt gibt es p–1 mögliche Prüfpaare (mit $p = |F^{(k)}|$; für jedes $b_j \neq 0$ eines mit $c_j := s + b_j y_j$), und alle sind gleich wahrscheinlich. Folglich ist die Wahrscheinlichkeit, daß E mit einem Authentikationsnachweis y'_j ein $s' \neq s$ akzeptiert, $\leq 1/(p-1)$.

Die gesuchte Fehlerwahrscheinlichkeit ist gerade die, daß E wenigstens eines der k Paare (s', y_j') akzeptiert. Die Wahrscheinlichkeit einer Disjunktion von Ereignissen kann man stets durch die Summe der Wahrscheinlichkeiten der einzelnen Ereignisse abschätzen (also $W(A_1 \vee \ldots \vee A_k) \leq W(A_1) + \ldots + W(A_k)$). Folglich ist die gesuchte Wahrscheinlichkeit

$$\leq \frac{k}{p-1} \leq \frac{2k}{p} \leq 2^{-k},$$

denn $\log(p) \geq k + \log_2(k) + 1$.

e) Die Polynomialität des Rechenaufwands ist offensichtlich. Alle Schritte in Phase 1 und Phase 2 benötigen jeweils eine eigene Runde. Der Nachrichtenaufwand im schlimmsten Fall wird jeweils durch die Vektoren dominiert: jeder solche benötigt $O(k^2)$ bit. ❒

2.1.2.3 RABINs Geheimnisfestlegungsprotokoll

Der nächste Schritt zur überprüften Geheimnisaufteilung ist RABINs **Geheimnisfestlegungsprotokoll** (*weak secret sharing*) [RaBe_89: WSS]. Dieses entsteht aus SHAMIRs Geheimnisaufteilung (Protokoll 2.1), indem der Geheimniskrämer G durch Authentikationsnachweise (Protokoll 2.2) sicherstellt, daß jeder Prozessor P_i jeden anderen P_j von der Authentizität seines Anteils β_i überzeugen kann.

Auf diese Weise legt sich G auf sein Geheimnis fest: Beim Zusammensetzen ergibt sich entweder das Geheimnis von G oder aber gar kein Geheimnis. Letzteres impliziert, daß G fehlerhaft ist.

Protokoll 2.3: RABINs Geheimnisfestlegungsprotokoll mit Parameter k

Der Geheimniskrämer G hat als lokale Eingabe sein Geheimnis $s \in F^{(k)}$.

Phase 1: Festlegen auf ein Geheimnis (d. h. Aufteilen von s)

G wählt zufällig ein Polynom

$$h \in \Pi_t(s)$$

und berechnet $\beta_i := h(\alpha_i)$ für alle $i = 1, \ldots, n$.

Für jedes Paar von Prozessoren (P_i, P_j) wird Phase 1 von Protokoll 2.2 (mit Parameter k) mit $(G, Z, E, s) = (G, P_i, P_j, \beta_i)$ ausgeführt. Der entsprechende Authentikationsnachweis sei $\boldsymbol{y_{ij}}$. (β_i kann dabei öffentlich werden.)

Phase 2: Aufdecken eines Geheimnisses

[1] G veröffentlicht das Polynom h und damit $s = h(0)$.

Jeder Prozessor P_i sendet seinen Anteil β_i, falls dieser nicht schon öffentlich ist, an jeden anderen Prozessor P_j, zusammen mit dem jeweiligen Authentikationsnachweis $\boldsymbol{y_{ij}}$.

[2] Jeder P_i entscheidet anhand seiner Prüfpaare für jedes erhaltene (β_j, $\boldsymbol{y_{ji}}$), ob er den Anteil β_j akzeptiert oder ablehnt. Wenn für einen öffentlichen Anteil β_j oder für einen Anteil β_j, den P_i akzeptiert hat, gilt $h(\alpha_j) \neq \beta_j$, so veröffentlicht P_i eine **Vetonachricht**.

[3] Wenn wenigstens t+1 Prozessoren Vetonachrichten veröffentlicht haben, so wird G disqualifiziert. Andernfalls wird $s = h(0)$ als das von G aufgeteilte Geheimnis **akzeptiert**.

Lemma 2.3: (RABINs Geheimnisfestlegungsprotokoll)
Ist $t < n/2$, so gilt:

a) Nach Phase 1 gilt:
   a1) Ist G korrekt, so erhalten die Störer keine Information über s.
   a2) Kein korrekter Prozessor wurde disqualifiziert.

b) Mit Wahrscheinlichkeit $\geq 1 - n^2 2^{-k}$ gelten folgende fünf Aussagen:
   b1) Wird G in Phase 1 nicht disqualifiziert, so werden die korrekten Prozessoren ihre Anteile gegenseitig akzeptieren.
   b2) Ist G korrekt, so werden in Phase 2 korrekte Prozessoren ausschließlich authentische Anteile akzeptieren.
   b3) Wird G in Phase 1 nicht disqualifiziert, so werden in Phase 2 alle korrekten Prozessoren darin übereinstimmen, G zu disqualifizieren oder $h(0)$ als Geheimnis zu akzeptieren. Akzeptieren sie $h(0)$, so liegen alle Anteile der korrekten Prozessoren und alle öffentlichen Anteile auf dem von G veröffentlichten Polynom h.
   b4) Kein korrekter Prozessor wird disqualifiziert.
   b5) Ist G korrekt, so akzeptieren in Phase 2 alle korrekten Prozessoren sein Geheimnis s.

c) Der Rechenaufwand des Protokolls ist polynomial in n und k.
   Phase 1 benötigt 6 Runden, in denen $O(n^2k^2)$ bit veröffentlicht und über sichere nichtabhörbare Verbindungen gesendet werden.
   Phase 2 benötigt 2 Runden, in denen $O(nk)$ bit veröffentlicht und $O(n^2k^2)$ bit über sichere Verbindungen gesendet werden. ♦

Lemma 2.3a,b entspricht [RaBe_89: Lemma 3, 4 und WSS-Theorem]. Die Aussagen und Beweise in [RaBe_89] sind allerdings unvollständig und die Fehlerbetrachtung

ungenau. Die Fehlerwahrscheinlichkeit von Lemma 2.3b4 wurde in [RaBe_89] ganz übersehen, d. h. dort wird behauptet, korrekte Prozessoren würden nie disqualifiziert [RaBe_89: Proof of the WSS Theorem, 1.].

Beweis:

a1) Nach Lemma 2.1a erhalten die höchstens $t < n/2$ Störer aus ihren Anteilen β_i keine Information über s, und nach Lemma 2.2b erhalten sie aus den Authentikationsnachweisprotokollen keine Information über die Anteile der korrekten Prozessoren.

a2) Lemma 2.2a.

b1, b2) Es wird danach unterschieden, ob G korrekt ist oder nicht.

Angenommen, G sei nicht korrekt. Dann gilt nach Lemma 2.2c für jedes der $< n^2$ korrekten Paare (Z, E) von Prozessoren mit Fehlerwahrscheinlichkeit $\leq 2^{-k}$, daß falls G nicht disqualifiziert wird, E den Anteil von Z akzeptiert. Folglich ist die Fehlerwahrscheinlichkeit für alle Paare zusammen $\leq n^2 2^{-k}$. Für (b2) ist nichts zu zeigen, d. h. die Fehlerwahrscheinlichkeit ist $\leq n^2 2^{-k}$.

Angenommen, G sei korrekt. Dann gilt (b1) fehlerfrei, denn sind G, Z und E korrekt, so akzeptiert E den Anteil von Z ohne jeden Fehler. Nach Lemma 2.2d gilt für jedes der $< n^2$ Paare (Z,E) aus fehlerhaftem Z und korrektem E mit Fehlerwahrscheinlichkeit $\leq 2^{-k}$, daß E von Z nur den authentischen Anteil akzeptiert. Folglich ist die Fehlerwahrscheinlichkeit für alle Paare zusammen, und damit auch die in diesem Fall, $\leq n^2 2^{-k}$.

b3) Folgt aus (b1) und (b2). Es wird unterschieden, ob G korrekt ist oder nicht.

Angenommen, G sei nicht korrekt. Da h und die Vetonachrichten zuverlässig verteilt werden, wählen alle korrekten Prozessoren in Phase 2, Schritt [3] dieselbe Alternative. Paßt ein öffentlicher Anteil oder der Anteil eines korrekten Prozessors nicht zu h, so haben dies wegen (b1) in Schritt [2] alle korrekten Prozessoren festgestellt und eine Vetonachricht gesendet. In diesem Fall erhalten alle wenigstens t+1 Vetonachrichten und akzeptieren h nicht.

Angenommen, G sei korrekt. Nach (b2) akzeptiert kein korrekter Prozessor einen nichtauthentischen Anteil, und da alle authentischen Anteile auf h liegen, veröffentlicht kein korrekter Prozessor eine Vetonachricht. Folglich wird G nicht disqualifiziert und alle korrekten Prozessoren akzeptieren $h(0)$.

b4) Für Phase 1 folgt dies aus (a2). In Phase 2 kann höchstens G disqualifiziert werden, und für G folgt es wie der zweite Fall im Beweis von (b3) aus (b2).

b5) Ist G korrekt, so wird er nach (b4) nicht disqualifiziert. Folglich akzeptieren in Phase 2, Schritt [3] alle korrekten Prozessoren $s = h(0)$.

c) Der erste Teil gilt offensichtlich.

Phase 1 besteht aus $O(n^2)$ parallelen Ausführungen von Protokoll 2.2, Phase 1.

Der Rest folgt aus Lemma 2.2e.

Phase 2 benötigt für die Schritte [1] und [2] je eine Runde, für Schritt [3] keine, insgesamt also 2. In Schritt [1] veröffentlicht G $O(nk)$ bit. Jeder P_i sendet seinen Anteil und die Authentikationsnachweise, also $O(k^2)$ bit, an jeden anderen; insgesamt sind dies $O(n^2k^2)$ bit. Schritt [2] spielt demgegenüber keine Rolle. ❒

In Phase 1 von RABINs Geheimnisfestlegungsprotokoll legt sich der Geheimniskrämer G auf einen Wert s fest, ohne den höchstens t Störern über diesen etwas zu verraten.

Am Ende von Phase 1 gibt es *keine* Entscheidung darüber, ob ein Geheimnis aufgeteilt wurde oder nicht. Ein fehlerhafter Geheimniskrämer kann daher im schlimmsten Fall verhindern, daß überhaupt ein Geheimnis aufgedeckt wird. Er kann jedoch fast sicher kein anderes als das in Phase 1 festgelegte Geheimnis aufdecken.

2.1.2.4 RABINs Protokoll zur überprüften Geheimnisaufteilung

Die überprüfte Geheimnisaufteilung stellt, im Unterschied zur Geheimnisfestlegung, auch sicher, daß nach Phase 1 die korrekten Prozessoren tatsächlich ein Geheimnis zusammensetzen können (falls G nicht disqualifiziert wurde). Das Zusammensetzen kann ohne Hilfe des Geheimniskrämers geschehen.

RABINs überprüfte Geheimnisaufteilung entsteht dadurch, daß jeder Prozessor sich auf seinen vom Geheimniskrämer erhaltenen Anteil festlegt. Die ganze Konstruktion wird mit einem *Zero-knowledge*-Beweis versehen, daß G tatsächlich ein Geheimnis aufgeteilt hat und daß die Geheimnisfestlegungen aller Prozessoren korrekt sind [RaBe_89: VSS].

Die Grundidee des *Zero-knowledge*-Beweises ist eine einfache *Cut-and-choose*-Prozedur [ChCD_88]

Zur Aufteilung muß G ein Polynom $f \in \Pi_t$ wählen und die Anteile $\beta_i = f(\alpha_i)$ verteilen. Damit das Zusammensetzen später gelingen kann, müssen die Anteile der korrekten Prozessoren alle zu einem Polynom vom Grade $\le t$ passen.

Dies wird anhand einer einfachen Eigenschaft von Polynomen nachgewiesen:

$$\text{Für alle Polynome } g \text{ gilt: } [\, g \in \Pi_t \text{ und } g+f \in \Pi_t \,] \Rightarrow f \in \Pi_t,$$

denn es ist $f = (g+f) - g$, und die Differenz zweier Polynome vom Grade $\le t$ hat selbst einen Grad $\le t$.

Zum Nachweis von „$f \in \Pi_t$“ wählt daher G unabhängig von f zufällig ein Polynom $g \in \Pi_t$ und verteilt die Anteile $\gamma_i := g(\alpha_i)$. Danach muß G entweder g oder $g' := g+f$ veröffentlichen; welches von beiden, wird zufällig entschieden. Jeder Prozessor P_i prüft, ob g bzw. g' einen Grad $\le t$ hat und zu γ_i bzw. $\beta_i + \gamma_i$ paßt. Falls nicht, so beschwert sich P_i. (P_i tut dasselbe auch für alle öffentlichen Anteile.)

Ist $f \notin \Pi_t$, so kann G nicht sowohl ein g als auch ein g' finden, so daß alle korrekten Prozessoren mit beiden zufrieden wären. Mit Wahrscheinlichkeit $\geq 2^{-1}$ beschwert sich daher wenigstens ein Prozessor. Durch μ-malige Iteration wird diese Wahrscheinlichkeit $\geq 1-2^{-\mu}$.

Die gesamte Prozedur wird solange wiederholt, bis sich entweder einmal in μ der obigen Miniprozeduren *kein* Prozessor über G beschwert hat, oder bis sicher ist, daß sich wenigstens *ein korrekter* Prozessor über G beschwert hat (also z. B. dann, wenn sich insgesamt t+1 Prozessoren beschwert haben). Im ersten Fall akzeptieren die korrekten Prozessoren, daß ihre Anteile β_i zu einem $f \in \Pi_t$ passen, im zweiten wird G disqualifiziert.

Um zu verhindern, daß sich ein Prozessor permanent beschwert, wird der Anteil β_i nach der ersten Beschwerde von P_i veröffentlicht. Dies schadet nichts, denn beschwert sich P_i, so ist entweder G oder P_i ein Störer, und β_i ist daher den Störern ohnehin schon bekannt.

(Analog wird auch nachgeprüft, ob sich alle Prozessoren auf ihre authentischen Anteile β_i festgelegt haben.)

Protokoll 2.4: RABINs Protokoll zur überprüften Geheimnisaufteilung mit Parametern k und m

Der Geheimniskrämer G hat als lokale Eingabe das Geheimnis $s \in F^{(k)}$.

Protokoll 2.3 wird jeweils mit Parameter k verwendet.

Phase 1: Aufteilen eines Geheimnisses

[1] G wählt zufällig ein Polynom

$$f \in \Pi_t(s)$$

und sendet $\beta_i := f(\alpha_i)$ an P_i, $i = 1, \ldots, n$.

[2] Jeder P_i verwendet Phase 1 von Protokoll 2.3, um sich gegenüber allen Prozessoren auf β_i festzulegen. Er verwendet hierfür ein zufällig gewähltes Polynom

$$h_i \in \Pi_t(\beta_i).$$

Verifikationsprozedur:

Die Korrektheit der Schritte [1] und [2] wird in höchstens t+1 Iterationen nachgeprüft. Der Iterationszähler w ist eine globale Variable und wird mit $w := 1$ initialisiert. Jede Iteration besteht aus bis zu n parallel ausgeführten Miniverifikationsprozeduren (jeweils nach dem *Cut-and-choose*-Prinzip). In der v-ten Miniverifikationsprozedur agiert P_V als Prüfer.

[3.*w*] *G* wählt zufällig *nm* Polynome

$$g_{v,u} \in \Pi_t$$

und berechnet die Anteile

$$\gamma_{v,u,i} := g_{v,u}(\alpha_i),$$

für $v = 1, \ldots, n$ und $u = 1, \ldots, m$ und alle Prozessoren P_i.

An jeden P_i, dessen Anteil β_i nicht öffentlich ist, sendet *G* die $\gamma_{v,u,i}$ für alle *u*, *v*. Ansonsten veröffentlicht *G* diese $\gamma_{v,u,i}$.

[4.*w*] Jeder P_i, dessen Anteil nicht öffentlich ist, legt sich auf alle in [3.*w*] erhaltenen $\gamma_{v,u,i}$ mit Phase 1 von Protokoll 2.3 fest. Er verwendet hierfür zufällig gewählte Polynome

$$h_{v,u,i} \in \Pi_t(\gamma_{v,u,i}).$$

P_i legt sich mit Phase 1 von Protokoll 2.3 auch auf die Werte

$$\delta_{v,u,i} := \beta_i + \gamma_{v,u,i}$$

fest, wofür er die Polynome

$$h_i + h_{v,u,i} \in \Pi_t(\beta_i + \gamma_{v,u,i})$$

verwendet. Die Anteile hieran *berechnet* jeder Prozessor P_j als Summe aus den ihm schon mitgeteilten Anteilen an β_j und $\gamma_{v,u,j}$. P_i verteilt daher lediglich die Authentikationsnachweise und Prüfpaare.

[5.*w*] Jeder P_v ($v = 1, \ldots, n$), der nicht disqualifiziert wurde, veröffentlicht für jedes $u \in \{1, \ldots, m\}$ eine aus {0,1} zufällig gewählte **Frage** $q_{v,u}$.

Ist $q_{v,u} = 0$, so veröffentlicht *G* das Polynom $g_{v,u}$, ansonsten $g_{v,u} + f$.

[a] Verifikation der öffentlichen Anteile: Für jeden öffentlichen Anteil β_j prüft jeder P_i, ob

$$\gamma_{v,u,j} = g_{v,u}(\alpha_j) \text{ bzw. } \beta_j + \gamma_{v,u,j} = (g_{v,u} + f)(\alpha_j)$$

gilt. Gilt eine dieser Gleichungen nicht, so wird *G* disqualifiziert.

[b] Verifikation von „*G* hat zusammenpassende Anteile verteilt“: Jeder P_i, dessen Anteil nicht öffentlich ist, prüft, ob

$$\gamma_{v,u,i} = g_{v,u}(\alpha_i) \text{ bzw. } \delta_{v,u,i} = (g_{v,u} + f)(\alpha_i)$$

gilt. Ist P_i nicht zufrieden, so veröffentlicht er eine **Vetonachricht**.

[c] Verifikation von „P_i hat sich auf β_i festgelegt“: Jeder P_i, dessen Anteil nicht öffentlich ist und der in [b] keine Vetonachricht gesendet hat, deckt mit Phase 2 von Protokoll 2.3

$$\gamma_{v,u,i} \text{ bzw. } \delta_{v,u,i}$$

auf. Wird der aufgedeckte Wert akzeptiert, liegt aber nicht auf $g_{v,u}$ bzw. $g_{v,u} + f$, so wird P_i disqualifiziert.

[6.*w*] Wenn in Schritt [5.*w*] keine Vetonachricht gesendet und *G* nicht disqualifiziert wurde, so **akzeptieren** alle korrekten Prozessoren, daß tatsächlich ein Geheimnis aufgeteilt wurde.

Andernfalls veröffentlicht *G* für jeden Prozessor P_i, der in Schritt [5.*w*] eine

Vetonachricht gesendet hat, den Anteil β_j.

Sei a_d die Anzahl der Prozessoren, die disqualifiziert wurden, und a_V die der nichtdisqualifizierten Prozessoren, die bislang eine Vetonachricht gesendet haben. Ist $a_d + a_V > t$, so wird G disqualifiziert. Ansonsten setzen alle Prozessoren $w := w+1$ und gehen zu Schritt [3.w].

Wurde G disqualifiziert, so wird die Geheimnisaufteilung von allen korrekten Prozessoren **abgelehnt**.

Phase 2: Zusammensetzen eines Geheimnisses

[1] Jeder P_i, dessen Anteil nicht öffentlich ist, verwendet Phase 2 von Protokoll 2.3, um β_i aufzudecken.

[2] Jeder P_i bildet die Menge S_i all der Anteile β_j, die er in Schritt [1] akzeptiert hat, und aller öffentlichen Anteile. Er interpoliert das Polynom f mit $t+1$ beliebigen Anteilen aus S_i und akzeptiert $f(0)$ als das Geheimnis s.

Anmerkung: Der bereits erwähnte Entwurfsfehler in [RaBe_89] bezieht sich auf die Verifikationsprozedur: Im Originalprotokoll werden die Schritte [3.w] bis [6.w] nur *einmal* ausgeführt. Hierdurch werden jedoch die *öffentlichen* Anteile nicht ausreichend überprüft (der Beweis von [RaBe_89: Lemma 7] erwähnt diese nie), wie ein einfaches Beispiel zeigt.

Beispiel: G und P_i sind fehlerhaft und verhalten sich in allen Schritten bis einschließlich Schritt [4.1] korrekt. In Schritt [5.1] sendet P_i eine Vetonachricht, worauf G in [6.1] einen *beliebigen* Wert $\beta_i' \neq f(\alpha_i)$ als Anteil von P_i veröffentlicht.

In Schritt [6.1] muß G in [RaBe_89] zusätzlich noch Werte $\gamma'_{v,u,i}$ veröffentlichen, anhand derer wie in Schritt [4.1] der Anteil β_i' überprüft werden soll. Dies ist jedoch sinnlos, denn G kennt nun schon alle Fragen $q_{v,u}$ und kann daher stets passende Werte $\gamma'_{v,u,i}$ veröffentlichen: War $q_{v,u} = 0$, so veröffentlicht G $\gamma'_{v,u,i} := g_{v,u}(\alpha_i)$, ansonsten $\gamma'_{v,u,i} := g_{v,u}(\alpha_i) + f(\alpha_i) - \beta_i'$.

β_i' wird folglich nicht mehr überprüft. In Phase 2 werden alle korrekten Prozessoren das falsche öffentliche β_i' akzeptieren und daher nicht mehr notwendigerweise alle das richtige Polynom f zusammensetzen.

Das Beispiel gilt auch dann, wenn alle nm Fragen $q_{v,u}$ sequentiell in nm Iterationen gestellt und beantwortet würden: P_i müßte mit seiner Vetonachricht dann nur bis zur letzten Iteration warten.

Das ursprüngliche Protokoll aus [RaBe_89] benötigt nur eine konstante Rundenanzahl, während die hier vorgestellte Verbesserung $O(t)$ Runden benötigt. (Vgl. aber Abschnitt 2.1.2.5.) ♦

Lemma 2.4: (RABINs Protokoll zur überprüften Geheimnisaufteilung)
Ist $t < n/2$, so gilt:

a) Ist G korrekt, so erhalten die Störer vor Phase 2 keine Information über s.

b) Mit Wahrscheinlichkeit $\geq 1 - (2mn^5 2^{-k} + n2^{-1-mn/2})$ gelten folgende Aussagen:

b1) Kein korrekter Prozessor wird disqualifiziert.

b2) Akzeptieren die korrekten Prozessoren in Phase 1, daß ein Geheimnis aufgeteilt wurde, so gibt es genau ein Geheimnis s, das die korrekten Prozessoren in Phase 2 zusammensetzen werden.

c) Der Rechenaufwand des Protokolls ist polynomial in k, m und n.

Phase 1 benötigt $10t + 17$ Runden, in denen $O(mn^5k^2)$ bit veröffentlicht und über sichere nichtabhörbare Verbindungen gesendet werden.

Phase 2 benötigt 2 Runden, in denen $O(n^2k)$ bit veröffentlicht und $O(n^3k^2)$ bit über sichere Verbindungen gesendet werden. ♦

Lemma 2.4 entspricht im Prinzip [RaBe_89: Lemma 5-7 und Main Theorem], allerdings ohne genauere Betrachtung von Fehlerwahrscheinlichkeit und Aufwand. Die Verifikationsprozedur wurde gegenüber [RaBe_89] korrigiert. Der Aussage von Lemma 2.4b2 entspricht inhaltlich [RaBe_89: Lemma 7 und Main Theorem]; beide sind aber in [RaBe_89] nur ansatzweise bewiesen und, wie oben bemerkt, so auch falsch.

Beweis:

a) Für die Schritte [1] bis [4.w] folgt dies aus Lemma 2.3a1.

In [5.w] wird für jedes Paar v, u genau eines der Polynome $g_{v,u}$ und $g'_{v,u} := g_{v,u}+f$ veröffentlicht und Anteile daran aufgedeckt. Beide allein enthalten keine Information über f: Für $g_{v,u}$ ist dies klar. Zu gegebenem $g'_{v,u}$ gibt es für jedes $f' \in \Pi_t$ genau ein Polynom $g_{v,u} \in \Pi_t$, das $g'_{v,u} = g_{v,u}+f'$ erfüllt. Da alle $g_{v,u} \in \Pi_t$ gleichwahrscheinlich sind, enthält $g'_{v,u}$ keine Information über f.

Hat P_i in [5.w] eine Vetonachricht veröffentlicht und ist G korrekt, so muß P_i fehlerhaft sein. Der Anteil von P_i ist den Störern daher ohnehin schon bekannt und kann in [6.w] gefahrlos veröffentlicht werden.

b) Vorbemerkung: Zum Beweis von (b) wird angenommen, für alle Ausführungen von Protokoll 2.3 träfen alle Aussagen von Lemma 2.3b zu.

Dies trifft jedoch nur mit Wahrscheinlichkeit

$$W_{Vor} \geq 1 - 2mn^5 2^{-k}$$

zu: Protokoll 2.3 wird in Phase 1, Schritt [2] höchstens n mal und in jedem Schritt [4.w] höchstens $2mn^2$ ausgeführt. Insgesamt wird Schritt [4.w] höchstens t+1 mal iteriert (denn spätestens dann wird G disqualifiziert). Also wird Protokoll 2.3 insgesamt höchstens (für $n \geq 1$)

$$n + 2(t+1)mn^2 \leq 2mn^3$$

mal ausgeführt. Damit trifft Lemma 2.3b mit Fehler $\leq 2mn^5 2^{-k}$ für alle diese

Ausführungen zu.

Die Fehlerwahrscheinlichkeit $\leq 2mn^5 2^{-k}$ muß später in die Fehlerwahrscheinlichkeit von (b) mit eingerechnet werden. ◊

b1) Für alle Prozessoren außer G folgt dies aus Lemma 2.3a2 und Lemma 2.3b4.

Angenommen, G wird in [6.w] disqualifiziert, d.h. es ist $a_d + a_V > t$. Da es höchstens t Störer gibt, muß sich unter den $a_d + a_V$ Prozessoren ein korrekter befinden, und da nach dem bisher gezeigten alle a_d disqualifizierten Prozessoren Störer sind, muß er sich unter den a_V sich beschwerenden Prozessoren befinden. Ein korrekter Prozessor veröffentlicht aber nur dann eine Vetonachricht, wenn zwei Informationen, die er beide von G erhalten hat, nicht zusammenpassen. Folglich ist G fehlerhaft.

b2) Zur Abkürzung sei folgende Notation eingeführt:

Wird in Schritt [5.w] nach einer Frage $q_{v,u}$ G nicht disqualifiziert und veröffentlicht kein korrekter Prozessor eine Vetonachricht (bzw. wird P_x nicht disqualifiziert), so hat G (bzw. P_x) die Frage $q_{v,u}$ **zufriedenstellend** beantwortet.

Zu einem aufgeteilten Geheimnis werden die Anteile der korrekten Prozessoren und die öffentlichen gemeinsam als die **korrekten Anteile** bezeichnet.

Um (b2) zu beweisen, wird bewiesen, daß mit entsprechender Wahrscheinlichkeit gilt:

G wird in Phase 1 disqualifiziert,

<u>oder</u> 1) alle korrekten Anteile liegen auf einem Polynom $f \in \Pi_t$, <u>und</u>

2) kein korrekter Prozessor wird in Phase 2 einen Anteil akzeptieren, der nicht auf dem Polynom liegt, das die korrekten Anteile definieren.

Ist dies gezeigt, so gilt auch (b2): Die korrekten Prozessoren akzeptieren, daß ein Geheimnis aufgeteilt wurde, genau dann, wenn G nicht disqualifiziert wird. Wird G nicht disqualifiziert, so werden wegen Lemma 2.3b5 die korrekten Prozessoren alle korrekten Anteile akzeptieren und genau das Polynom f erhalten.

Gesucht ist nun die Wahrscheinlichkeit, mit der das obige Ereignis eintritt, unter der Bedingung, daß die Vorbemerkung zutrifft.

Man betrachte zunächst eine Iteration der Verifikationsprozedur und eine Frage $q_{v,u}$, die von einem korrekten Prozessor P_v stammt und daher aus {0,1} zufällig gewählt wird. Sei f^* das Polynom kleinsten Grades, zu dem alle Anteile der korrekten Prozessoren und alle bis dahin öffentlichen Anteile passen (d.h. das durch sie interpoliert wird).

Angenommen,

B_1) $f^* \notin \Pi_t$, oder

B_2) es gäbe einen Störer P_x, dessen Anteil noch nicht öffentlich ist, von dem aber in Phase 2 ein korrekter Prozessor P_k einen Anteil β_x^* akzeptieren

würde, der nicht zu f^* paßt (dies ist hier ja bereits festgelegt).
Dann wird aufgrund dieser Frage mit Wahrscheinlichkeit $\geq 1/2$ gelten: G ist disqualifiziert oder wenigstens ein Prozessor sendet eine Vetonachricht.

Grund:

i) Angenommen, (B_1) trifft zu, d. h. $f^* \notin \Pi_t$.

Dann kann G nicht *beide* möglichen Fragen $q_{v,u} \in \{0,1\}$ zufriedenstellend beantworten: Angenommen, G könnte es doch, d. h. es gäbe Polynome $g_{v,u}, g'_{v,u} \in \Pi_t$, so daß alle korrekten Anteile $\gamma_{v,u,i}$ zu $g_{v,u}$ und alle korrekten Anteile $\gamma_{v,u,i} + \beta_i$ zu $g'_{v,u}$ passen. Dann würden aber auch alle korrekten Anteile β_i zu $g'_{v,u} - g_{v,u}$ passen, d. h. es wäre $f^* = g'_{v,u} - g_{v,u} \in \Pi_t$, im Widerspruch zur Annahme.

Da $q_{v,u}$ in $\{0,1\}$ gleichverteilt ist, wird P_v mit Wahrscheinlichkeit $\geq 1/2$ eine Frage auswählen, die G nicht korrekt beantworten kann. In diesem Fall wird entweder G disqualifiziert, oder aber wenigstens ein korrekter Prozessor sendet eine Vetonachricht.

ii) Angenommen, (B_1) trifft nicht zu (d. h. $f^* \in \Pi_t$), aber (B_2) trifft zu.

Dann können nicht G und P_x *beide* möglichen Fragen $q_{v,u} \in \{0,1\}$ zufriedenstellend beantworten: Wieder angenommen, sie könnten es doch. Dann gibt es Polynome $g_{v,u}, g'_{v,u} \in \Pi_t$, so daß alle korrekten Anteile $\gamma_{v,u,i}$ zu $g_{v,u}$ und alle korrekten Anteile $\gamma_{v,u,i} + \beta_i$ zu $g'_{v,u}$ passen. Es ist dann (s. o.)

$$f^* = g'_{v,u} - g_{v,u}.$$

Nach Annahme beschwert sich P_x in [b] nicht. Damit er in [c] nicht disqualifiziert wird, muß er daher Anteile $\gamma_{v,u,x}$ und $\delta_{v,u,x}$ aufdecken können, die zu $g_{v,u}$ und $g'_{v,u}$ passen. Ist dies der Fall, so paßt aber die Differenz

$$\beta_x := \delta_{v,u,x} - \gamma_{v,u,x}$$

zu f^*; und da alle korrekten Prozessoren ihre Anteile an $\delta_{v,u,x}$ lokal als Summe ihrer Anteile an $\gamma_{v,u,x}$ und dem in Schritt [2] festgelegten Geheimnis berechnet haben, haben sie in Schritt [2] alle Anteile erhalten, die zu dem korrekten Anteil β_x passen. Wegen der Vorbemerkung und Lemma 2.3b3 wird daher kein korrekter Prozessor später ein $\beta_x^* \neq \beta_x$ akzeptieren, im Widerspruch zur Annahme. ◊

Es gibt insgesamt wenigstens $n-t \geq n/2$ korrekte nichtdisqualifizierte Prozessoren (b1), von denen jeder in einer Verifikationsprozedur m Fragen stellt. Damit sind $(n-t)m \geq nm/2$ Fragen zufällig gestellt. In einer Iteration der Verifikationsprozedur gilt daher mit Wahrscheinlichkeit $\geq 1 - 2^{-nm/2}$ (mit den Bedingungen (B_i) wie oben):

Gilt zu Beginn der Iteration (B_1) oder (B_2), so gilt nach dieser Iteration mit Wahrscheinlichkeit $\geq 1 - 2^{-nm/2}$: G ist disqualifiziert oder wenigstens ein Prozessor sendete eine Vetonachricht.

Die Verifikationsprozedur wird solange wiederholt, bis entweder G disqualifiziert ist oder aber einmal keine Vetonachricht gesendet wurde. G wird spätestens nach t+1 Vetonachrichten disqualifiziert.

Damit die Aussage von (b2) nicht zutrifft, dürfen daher höchstens t Vetonachrichten gesendet werden. Die Wahrscheinlichkeit hierfür ist sicher höchstens die, daß bei t Iterationen wenigstens einmal das Ereignis eintritt, daß weder G disqualifiziert wird, noch daß irgendein Prozessor eine Vetonachricht sendet. Die Wahrscheinlichkeit hierfür ist wieder

$$\leq t\, 2^{-nm/2} \leq n\, 2^{-1-nm/2}.$$

Damit kann man nun die Fehlerwahrscheinlichkeit für (b) zusammentragen:

Für (b1) genügte die Vorbemerkung, d. h. es ist

$$\begin{aligned} \mathrm{W}((\mathrm{b1}) \wedge (\mathrm{b2})) &\geq \mathrm{W}(\text{Vorbemerkung} \wedge (\mathrm{b2})) = W_{Vor} \cdot \mathrm{W}((\mathrm{b2}) \mid \text{Vorbemerkung}) \\ &\geq (1 - 2mn^5 2^{-k})\,(1 - n\, 2^{-1-nm/2}) \geq 1 - 2mn^5 2^{-k} - n\, 2^{-1-nm/2} \end{aligned}$$

c) In jeder Iteration erhöht sich a_d+a_V, also gibt es höchstens t+1 Iterationen.

Der erste Teil gilt daher wieder offensichtlich.

Phase 1:

[1] Eine Runde, in der n Werte aus $F^{(k)}$, also $O(nk)$ bit über sichere, nichtabhörbare Verbindungen gesendet werden.

[2] n Ausführungen von Protokoll 2.3, Phase 1, parallel. Nach Lemma 2.3c: 6 Runden, in denen $O(n^3k^2)$ bit veröffentlicht und über sichere nichtabhörbare Verbindungen gesendet werden.

[3.w] Parallel mn mal Schritt [1], also: Eine Runde, in der $O(mn^2k)$ bit über sichere, nichtabhörbare Verbindungen gesendet werden.

[4.w] $O(n^2m)$ Ausführungen von Protokoll 2.3, Phase 1, parallel. Nach Lemma 2.3c: 6 Runden, in denen $O(mn^4k^2)$ bit veröffentlicht und über sichere nichtabhörbare Verbindungen gesendet werden.

[5.w] Für jedes (u, v), also mn mal (parallel):

1. Runde: P_v veröffentlicht in einer Runde 1 bit.
2. Runde: G veröffentlicht ein Polynom aus Π_t, also $O(nk)$ bit.
3. Runde: $\leq n$ Prozessoren veröffentlichen ein Veto, also $O(n)$ bit.

Danach wird für bis zu n Werte Phase 2 von Protokoll 2.3 ausgeführt. Nach Lemma 2.3c benötigt dies 2 Runden, in denen $O(n^2k)$ bit veröffentlicht und $O(n^3k^2)$ bit über sichere Verbindungen gesendet werden. Die beiden Runden können aber mit den obigen Runden 2 bzw. 3 verschmolzen werden.

Insgesamt sind dies daher 3 Runden, in denen $O(mn^3k)$ bit veröffentlicht und $O(mn^4k^2)$ bit über sichere, nichtabhörbare Verbindungen versendet werden.

[6.w] Muß die Iteration wiederholt werden, so veröffentlicht G bis zu t

Anteile, also in einer Runde $O(nk)$ bit. Dies kann aber parallel zu [3.w+1] geschehen, benötigt also keine eigene Runde.

Für alle Schritte [x.w] zusammen ergeben sich 10 Runden, in denen $O(mn^4k^2)$ bit veröffentlicht und über sichere nichtabhörbare Verbindungen gesendet werden. Da es höchstens t+1 Iterationen gibt, sind dies insgesamt $10t$+10 Runden, in denen $O(mn^5k^2)$ bit veröffentlicht und über sichere nichtabhörbare Verbindungen gesendet werden.

Für alle Runden zusammen erhält man daher im schlimmsten Fall die angegebenen Werte.

Phase 2: Nur in Schritt [1] werden Nachrichten gesendet: Phase 2 von Protokoll 2.3 wird höchstens n mal parallel ausgeführt. ❒

Die Rundenanzahl von Phase 1 kann durch weiteres Zusammenfassen von Runden auf $4t$+11 Runden reduziert werden:

Bis auf die Veröffentlichung der $\gamma_{v,u,i}$ für öffentliche Anteile kann man jeden Schritt [3.w] parallel zu [1] ausführen, und jeden Schritt [4.w] parallel zu [2]. Die Verifikationsprozedur wird höchstens t+1 mal ausgeführt, d. h. es genügt für das gesamte Protokoll, wenn für alle $w \in \{1, \ldots, t+1\}$ parallel zu Schritt [1] bzw. [2] die Schritte [3.w] soweit möglich bzw. die Schritte [4.w] ausgeführt werden. In jeder Iteration muß man dann statt [3.w] und [4.w] nur eine Runde vorsehen, in der die $\gamma_{v,u,i}$ für öffentliche Anteile veröffentlicht werden. Die Veröffentlichung von Anteilen in [6.w] muß statt mit [3.w] mit der ersten Runde von [5.w] zusammengefaßt werden. Dies spart in der Schleife jeweils 6 Runden. Die Gesamtrundenanzahl ist daher $7 + 4(t+1) = 4t + 11$.

Wird die Verifikationsprozedur tatsächlich t+1 mal iteriert, so erhöht sich der Nachrichtenaufwand durch diese Änderung nicht. Wird die Verifikationsprozedur seltener iteriert, so wurden [3.w] und [4.w] unnötig oft ausgeführt. Wird Protokoll 2.4 mit demselben Geheimniskrämer mehrfach ausgeführt, so kann man allerdings die „unverbrauchten“ und von f unabhängigen Teile von [3.w] und [4.w] (d. h. die unverbrauchten Polynome $g_{v,u}$, $h_{v,u,i}$, die Anteile hieran und die zugehörigen Authentikationsnachweise und Prüfpaare) für die nächste Ausführung von Protokoll 2.4 aufbewahren.

2.1.2.5 Einsparung von Iterationen der Verifikationsprozedur im Protokoll zur überprüften Geheimnisaufteilung

In Abschnitt 4.3 wird der Fall betrachtet, daß Phase 1 von Protokoll 2.4 sequentiell u mal ausgeführt wird. Nach Lemma 2.4c werden dafür $O(ut)$ Runden benötigt.

Man kann aber sehr leicht ohne Erhöhung des Aufwands erreichen, daß für u Ausführungen $O(u+n^2)$ Runden genügen. Für $u = n^c$, $c \geq 2$, und hinreichend

große n benötigt das Protokoll dann im Mittel wieder eine konstante Rundenanzahl.

Die eigentliche Verifikationsprozedur in Protokoll 2.4 benötigt nur eine konstante Rundenanzahl. Die O(t) Runden werden dadurch erreicht, daß in einer Protokollausführung die Verifikationsprozedur bis zu t+1 mal wiederholt wird.

Eine Wiederholung findet dann statt, wenn eine Vetonachricht gesendet wird. Hat ein Prozessor, etwa P_i, eine Vetonachricht gesendet und ist P_j der gerade aktuelle Geheimniskrämer, so können nicht sowohl P_i als auch P_j korrekt sein. Es schadet daher nichts, wenn in *allen* folgenden Ausführungen von Protokoll 2.4, in denen P_i (bzw. P_j) als Geheimniskrämer agiert, P_i (bzw. P_j) die Anteile für P_j (bzw. P_i) veröffentlicht. Tut man dies, so können P_i und P_j gegeneinander höchstens einmal eine Vetonachricht senden. Da es weniger als $n^2/2$ Paare (P_i, P_j) gibt, können insgesamt höchstens $n^2/2$ Vetonachrichten gesendet werden.

Die Verifikationsprozedur in Protokoll 2.4 wird daher für beliebig viele sequentielle Ausführungen höchstens $n^2/2$ mal iteriert. Für u sequentielle Ausführungen werden damit weniger als $17u + 5n^2$ Runden benötigt.

Dieselbe Einsparung von Iterationen ist auch für das modifizierte Protokoll 2.6 möglich.

2.1.3 Überprüfte Geheimnisaufteilung ohne zuverlässigen Verteilkanal beim Zusammensetzen

Im folgenden wird RABINs Protokoll zur überprüften Geheimnisaufteilung (Protokoll 2.4) so modifiziert, daß nur noch zum Aufteilen, nicht aber zum Zusammensetzen ein zuverlässiger Verteilkanal benötigt wird.

Dies wird es in Abschnitt 2.2 ermöglichen, innerhalb eines Verteilungsprotokolls Geheimnisse aufzudecken, die zuvor in einem Vorbereitungsprotokoll mit zuverlässigem Verteilkanal aufgeteilt wurden.

In Abschnitt 2.1.3.1 wird zunächst eine Abschwächung von RABINs Geheimnisfestlegung beschrieben und auf ihr aufbauend in Abschnitt 2.1.3.2 ein Protokoll zur überprüften Geheimnisaufteilung. Beide benötigen nur zum Aufteilen eines Geheimnisses einen zuverlässigen Verteilkanal. In Abschnitt 2.1.3.3 werden die freien Parameter der Protokolle so bestimmt, daß ihre Fehlerwahrscheinlichkeit jeweils $\leq 2^{-\tau}$ für ein frei wählbares τ ist.

2.1.3.1 Schwache Geheimnisfestlegung

Im folgenden wird eine Modifikation von RABINs Protokoll zur Geheimnisfestlegung (Protokoll 2.3, Abschnitt 2.1.2.3) beschrieben, genannt **schwache**

Geheimnisfestlegung. Sie benötigt nur zum Festlegen auf ein Geheimnis, nicht jedoch zum Aufdecken, einen zuverlässigen Verteilkanal.

Die Modifikation erreicht fast dasselbe wie das ursprüngliche Protokoll 2.3. Der einzige Unterschied ist, daß ein fehlerhafter Geheimniskrämer erreichen kann, daß einige korrekte Prozessoren ein Geheimnis akzeptieren (welches dann das richtige ist) und zugleich einige andere den Geheimniskrämer lokal disqualifizieren.

Protokoll 2.5: Protokoll zur schwachen Geheimnisfestlegung mit Parameter k

Phase 1: Festlegen auf ein Geheimnis

Wie Phase 1 von Protokoll 2.3 (RABINs Geheimnisfestlegungsprotokoll)

Phase 2: Zusammensetzen eines Geheimnisses

[1] Jeder Prozessor P_i sendet seinen Anteil β_i, falls dieser nicht schon öffentlich ist, an jeden anderen Prozessor P_j, zusammen mit dem jeweiligen Authentikationsnachweis $\boldsymbol{y_{ij}}$. (Für diesen Schritt wird keine zuverlässige Verteilung vorausgesetzt!)

[2] Jeder P_i entscheidet anhand seiner Prüfpaare für jedes erhaltene $(\beta_j, \boldsymbol{y_{ji}})$, ob er den Anteil β_j akzeptiert oder ablehnt.

[3] Wenn P_i wenigstens t+1 Anteile β_j, die öffentlichen und seinen eigenen eingeschlossen, akzeptiert, so interpoliert er alle akzeptierten Anteile mit einem Polynom h'. Ist $h' \notin \Pi_t$, so disqualifiziert P_i lokal den Geheimniskrämer.

Hat P_i den Geheimniskrämer G nicht disqualifiziert, so **akzeptiert** er $s = h'(0)$ als das Geheimnis von G.

Protokoll 2.5 wird in Abschnitt 2.2.1 in einer Situation eingesetzt, in der zwar alle korrekten Prozessoren die Schritte [2] bis [3] von Phase 2 ausführen, nicht jedoch auch Schritt [1]. Aus diesem Grunde ist der folgende Satz 2.5 etwas komplizierter formuliert, als es hier nötig erscheint.

Satz 2.5: (Protokoll zur schwachen Geheimnisfestlegung)
Ist $t < n/2$, so gilt:

a) Nach Phase 1 gilt:
   a1) Ist G korrekt, so erhalten die Störer keine Information über s.
   a2) Kein korrekter Prozessor wurde disqualifiziert.

b) Mit Wahrscheinlichkeit $\geq 1 - n^2 2^{-k}$ gelten folgende vier Aussagen:
   b1) Wird G in Phase 1 nicht disqualifiziert, so werden die korrekten Pro-

zessoren ihre Anteile gegenseitig akzeptieren.

b2) Ist G korrekt und akzeptiert ein korrekter Prozessor P_i wenigstens t+1 Anteile, so setzt er das korrekte Geheimnis s zusammen.

b3) Ist G korrekt und nehmen alle korrekten Prozessoren an Schritt [1] von Phase 2 teil, so setzen sie das korrekte Geheimnis s zusammen.

b4) Angenommen, G wird in Phase 1 nicht disqualifiziert und es gibt nach Phase 1 ein Polynom $h \in \Pi_t$, auf dem alle Anteile der korrekten Prozessoren und die öffentlichen Anteile liegen. Nehmen alle korrekten Prozessoren an Schritt [1] von Phase 2 teil, so akzeptiert jeder korrekte Prozessor P_i, der G nicht lokal disqualifiziert, $s = h(0)$. Gibt es kein solches h, so disqualifiziert jeder korrekte Prozessor lokal G.

c) Der Rechenaufwand des Protokolls ist polynomial in n und k. Phase 2 benötigt eine Runde, in der $O(n^2k^2)$ bit über sichere, nicht notwendigerweise nichtabhörbare Verbindungen, gesendet werden ♦

Satz 2.5b4 gilt für jeden Geheimniskrämer, auch für fehlerhafte, und beschreibt den Fall, daß sich der Geheimniskrämer G zwar in Phase 1, nicht aber notwendigerweise in Phase 2 korrekt verhält. Dies wird im Zusammenhang mit überprüfter Geheimnisaufteilung wichtig sein, denn dort wird der Geheimniskrämer durch die Verifikationsprozedur gezwungen, sich in Phase 1 korrekt zu verhalten.

Beweis:

a) Identisch mit Lemma 2.3a, denn Phase 1 hat sich gegenüber Protokoll 2.3 nicht geändert.

b) Da Phase 1 gegenüber Protokoll 2.3 nicht verändert wurde, und jeder Prozessor in Phase 2 dieselben Anteile akzeptiert wie in Protokoll 2.3, gilt Lemma 2.3b, d. h. mit Wahrscheinlichkeit $\geq 1 - n^2 2^{-k}$ gilt:

b1) Lemma 2.3b1.

b2) Nach Lemma 2.3b2 akzeptiert P_i nur authentische Anteile, und da G korrekt ist, definieren beliebige t+1 authentische Anteile s.

b3) Nach (b1) akzeptiert jeder korrekte Prozessor die Anteile eines jeden anderen, d. h. hat t+1 Anteile, und setzt nach (b2) das korrekte s zusammen.

b4) Nach (b1) akzeptieren alle korrekten Prozessoren die wenigstens t+1 Anteile der korrekten Prozessoren und die öffentlichen. Folglich ist h das einzige Polynom vom Grade t, zu dem alle Anteile passen, und jeder korrekte Prozessor, der überhaupt ein Polynom akzeptiert, akzeptiert dieses h.

c) Phase 1 ist unverändert, Phase 2 sogar kürzer als in Protokoll 2.3, so daß der erste Teil offensichtlich ist. In Phase 2 werden nur in Schritt [1] in einer Runde Nachrichten gesendet: jeder P_i sendet an jeden anderen einen Anteil ($O(k)$ bit)

und Authentikationsnachweise, also $O(k^2)$ bit. ❐

2.1.3.2 Überprüfte Geheimnisaufteilung

Das folgende Protokoll zur überprüften Geheimnisaufteilung benötigt keinen zuverlässigen Verteilkanal beim Zusammensetzen eines Geheimnisses. Es wird bewiesen, daß es ansonsten dieselben Eigenschaften hat wie RABINs Protokoll.

Protokoll 2.6: Überprüfte Geheimnisaufteilung ohne zuverlässigen Verteilkanal beim Zusammensetzen mit Parametern *k* und *m*

Phase 1: Aufteilen eines Geheimnisses

Wie Phase 1 von Protokoll 2.4. (Insbesondere wird dort, wo Geheimnisse *aufgedeckt* werden, Protokoll 2.3 verwendet, nicht Protokoll 2.5.)

Phase 2: Zusammensetzen eines Geheimnisses

[1] Jeder P_i nimmt am Zusammensetzen des Anteils β_j jedes anderen Prozessors P_j mittels Phase 2 des Protokolls 2.5 zur schwachen Geheimnisfestlegung teil, falls β_j nicht ohnehin schon öffentlich ist.

[2] Jeder P_i bildet die Menge S_i all der Anteile β_j, die er in Schritt [1] akzeptiert hat, und aller öffentlichen Anteile. Jeder P_i nimmt beliebige t+1 Anteile aus S_i, interpoliert das Polynom f und akzeptiert $f(0)$ als das Geheimnis s.

Satz 2.6: (Überprüfte Geheimnisaufteilung)

Ist $t < n/2$, so gilt:

a) Ist G korrekt, so erhalten die Störer vor Phase 2 keine Information über s.

b) Mit Wahrscheinlichkeit $\geq 1 - (2mn^5 2^{-k} + n\, 2^{-1-mn/2})$ gelten folgende Aussagen:

b1) Kein korrekter Prozessor wird disqualifiziert.

b2) Akzeptieren die korrekten Prozessoren in Phase 1, daß ein Geheimnis aufgeteilt wurde, so gibt es genau ein Geheimnis s, das die korrekten Prozessoren in Phase 2 zusammensetzen werden.

c) Der Rechenaufwand des Protokolls ist polynomial in k, m und n.

Phase 2 benötigt eine Runde, in der $O(n^3k^2)$ bit über sichere, nicht notwendigerweise nichtabhörbare, Verbindungen gesendet werden. ♦

Beweis:

a) Identisch mit Lemma 2.4a, denn Phase 1 hat sich gegenüber Protokoll 2.4 nicht geändert.

b) Wird gezeigt wie in Lemma 2.4b, d. h. unter derselben Bedingung wie dort bezüglich des Zutreffens von Lemma 2.3b bzw. nun auch Satz 2.5b (vgl. die dortige Vorbemerkung).

b1) Folgt für Phase 1 aus Lemma 2.4b1. In Phase 2 werden hier keine Prozessoren disqualifiziert.

b2) Nach Satz 2.5b1 akzeptieren alle korrekten Prozessoren die wenigstens $t+1$ Anteile der korrekten Prozessoren und die öffentlich bekannten. Folglich ist $|S_i| \geq t+1$ für jeden korrekten P_i, d. h. jeder P_i wird ein Geheimnis zusammensetzen.

Um zu beweisen, daß dies auch das richtige ist, wird gezeigt, daß, wenn *ein* korrekter Prozessor P_i hier einen Anteil β_x eines Störers P_x akzeptiert, *alle* korrekten Prozessoren β_x in Protokoll 2.4 (mit derselben Phase 1) akzeptieren könnten. (b2) folgt dann aus Lemma 2.4b2, d. h. β_x muß zu den $t+1$ Anteilen der korrekten Prozessoren und den öffentlich bekannten Anteilen passen.

Zum Beweis betrachte man ein beliebiges Störprotokoll X für Protokoll 2.6. Aus diesem soll ein Störprotokoll X' für Protokoll 2.4 entstehen, das dazu führt, daß in Protokoll 2.4 β_x von allen akzeptiert wird:

Die Phasen 1 beider Protokolle sind identisch, und dies gilt auch für die Störprotokolle X und X'.

Die Phasen 2 der Protokolle unterscheiden sich nur darin, mit welchem Unterprotokoll Anteile β_x akzeptiert werden, Protokoll 2.3 oder Protokoll 2.5. In Protokoll 2.3 kann, falls P_x ein Störer ist, X' in Phase 2, Schritt [1] ein Polynom h_x veröffentlichen und in Schritt [2] Vetonachrichten senden. Letzteres tue X' nie. Haben in Phase 1 alle korrekten Prozessoren Anteile erhalten, die zu einem Polynom $h_x \in \Pi_t(\beta_x)$ passen, so veröffentlicht X' in Protokoll 2.3, Phase 2, Schritt [1] dieses h_x. Sonst senden die Störer nichts bzgl. β_x (auch keine Anteile).

Angenommen, in Protokoll 2.5 akzeptiert ein korrekter P_i einen Anteil β_x eines Störers P_x.

Da vorausgesetzt wird, Phase 1 habe fehlerfrei gearbeitet, akzeptiert P_i die Anteile aller korrekten Prozessoren, und da er β_x akzeptiert, gibt es ein Polynom $h_x \in \Pi_t(\beta_x)$, das zu all diesen Anteilen paßt. Dieses h_x veröffentlicht P_x in Protokoll 2.3 mit obigem Störprotokoll X'. In Protokoll 2.3 wird dann kein Prozessor eine Vetonachricht senden (denn alle korrekten Anteile passen zu h_x), und folglich wird in Protokoll 2.3 β_x von allen korrekten Prozessoren akzeptiert.

c) Der erste Teil gilt offensichtlich. In Phase 2 werden nur in Schritt [1] Nachrichten gesendet: Bis zu n mal wird Phase 2 von Protokoll 2.5 ausgeführt. Nach Satz 2.5c erfordert dies eine Runde, in der jeder korrekte Prozessor an jeden anderen $O(nk^2)$ bit sendet, insgesamt also $O(n^3k^2)$ bit. ❐

2.1.3.3 Wahl der Protokollparameter

Sei τ ein Wert, so daß für die beiden Protokolle dieses Abschnitts 2.1.3 jeweils eine Fehlerwahrscheinlichkeit $\leq 2^{-\tau}$ akzeptabel ist. Da Protokoll 2.5 in Protokoll 2.6 enthalten ist, muß seine Fehlerwahrscheinlichkeit nicht getrennt betrachtet werden.

Nach Satz 2.6 hat Protokoll 2.6 einen Fehler $\leq 2mn^5 2^{-k} + n2^{-1-mn/2}$. Durch Wahl von m, k kann man beide Summanden $\leq 2^{-\tau-1}$ wählen:

$$2mn^5 2^{-k} \leq 2^{-\tau-1} \Leftrightarrow k \geq \tau + \log_2(m) + 5\log_2(n) + 2$$

$$n\,2^{-1-mn/2} \leq 2^{-\tau-1} \Leftrightarrow m \geq \frac{2\tau + 2\log_2(n)}{n}$$

Im folgenden wird daher angenommen, daß für die Fehlerwahrscheinlichkeit $2^{-\tau}$ die Werte k, m wie folgt gewählt werden:

$$m := \left\lceil \frac{2\tau + 2\log_2(n)}{n} \right\rceil,$$

$$k := \lceil \tau + \log_2(m) + 5\log_2(n) + 2 \rceil.$$

$F^{(k)}$ wird wie in Abschnitt 2.1.2.2 definiert gewählt, d. h.

$$F^{(k)} := \mathrm{GF}(2^{u_k}) \text{ mit } u_k := \lceil k + \log_2(k) + 1 \rceil.$$

Der obere Index von $F^{(k)}$ wird im folgenden nicht mehr geschrieben.

Mit den hier gemachten Festlegungen ist $m = O(\tau)$ (denn $\log_2(n)/n$ ist beschränkt) und $k = O(\tau + \log(n))$. Damit sind die Rechenaufwände der Protokolle dieses Kapitels jeweils polynomial in τ und n.

Beispiel: Verlangt man $\tau = 100$ und nimmt $n = 128$ an, so genügt $m = 2$, $k = 138$ und $F^{(138)} = \mathrm{GF}(2^{147})$. Für $n = 5$ sind $m = 41$, $k = 119$ und $F^{(119)} = \mathrm{GF}(2^{127})$ ausreichend. Für größere Werte n wird m kleiner (denn $\ln(n)/n$ fällt monoton für $n \geq e$), k wächst nur logarithmisch in n. ◣

2.2 Byzantinische Verteilung mit korrekter Mehrheit

Im folgenden werden drei Protokolle zur binären byzantinischen Verteilung beschrieben, die jede Anzahl $t < n/2$ tolerieren.

Das Protokoll in Abschnitt 2.2.1 setzt direkt auf den in Abschnitt 2.1.3 beschriebenen Protokollen zur Geheimnisaufteilung auf (dem eiligen Leser werden aber die Schlußbemerkungen aus Abschnitt 2.1.1 genügen) und erreicht damit in $O(L)$ Runden eine Fehlerwahrscheinlichkeit $\approx 1/L$. Durch Iteration dieses Protokolls wird in Abschnitt 2.2.2 in $O(\sigma)$ Runden eine Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ erreicht. Durch Einführung eines Abbruchkriteriums wird hieraus in Abschnitt 2.2.3 ein Protokoll mit Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ gewonnen, das im schlimmsten Fall denselben Aufwand verursacht wie das in Abschnitt 2.2.2, aber in erwartet weniger als 19 Runden terminiert.

Die drei folgenden Protokolle sind die ersten, die das byzantinische Verteilungsproblem für ein $t \geq n/3$ im byzantinischen Fehlermodell lösen.

2.2.1 Byzantinische Verteilung mit Fehlerwahrscheinlichkeit $\approx 1/L$ in $O(L)$ Runden

Das folgende Protokoll 2.7 löst das Problem der binären byzantinischen Verteilung mit einer linear kleinen Fehlerwahrscheinlichkeit.

Es benötigt ein Vorbereitungsprotokoll, zu dessen Ausführung ein zuverlässiger Verteilkanal und sichere nichtabhörbare Verbindungen vorausgesetzt werden. Das eigentliche Verteilungsprotokoll benötigt nur sichere Verbindungen.

Das Verteilungsprotokoll besteht aus L Iterationen eines einfachen 2-Runden-Protokolls, an dessen Ende jeder Prozessor P_i jeweils einen vorläufigen Endwert $v_i^{(l)}$ ($l = 0, \ldots, L-1$) bestimmt. (In Protokoll 2.7 bilden jeweils [l,b] und [l+1,a] ein solches 2-Runden-Protokoll; in [0,a] wird nur der Wert des Transmitters verteilt.) Aus allen L vorläufigen Entscheidungen wird dann am Ende des Protokolls zufällig eine ausgewählt. Grund hierfür ist, daß von den L vorläufigen Entscheidungen höchstens eine inkonsistent ist. Wird am Ende nicht gerade diese inkonsistente Entscheidung ausgewählt, so löst das Protokoll das byzantinische Verteilungsproblem. Die Fehlerwahrscheinlichkeit ist daher $\approx 1/L$.

In Abschnitt 2.2.2 und Abschnitt 2.2.3 wird Protokoll 2.7 mit $L = 3$ bzw. $L = 2$ verwendet.

Der zur Auswahl der endgültigen Entscheidung aus den L vorläufigen benötigte **Münzwurf** wird mit Hilfe von Protokoll 2.6, der überprüften Geheimnisaufteilung, realisiert: Im Vorbereitungsprotokoll teilt jeder Prozessor P_i einen zufällig gewählten Wert $r_i \in \{0, \ldots, L-1\}$ auf. Werden im Verteilungsprotokoll

dann die einzelnen r_i zusammengesetzt, so ist die Summe $R := r_1 + \ldots + r_n$ ein in $\{0, \ldots, L-1\}$ fast gleichverteilter und vor der Zusammensetzung der r_i nicht vorhersagbarer Wert.

Damit die Werte $0, \ldots, L-1$ aufgeteilt werden können, müssen sie durch Elemente aus F (dem für die Protokolle 2.5 und 2.6 verwendeten endlichen Körper) dargestellt werden. Dies geschieht mit Hilfe zweier Abbildungen ι und ι^{-1}: ι bildet $l \in \{0, \ldots, L-1\}$ injektiv in F ab; ι^{-1} bildet F auf $\{0, \ldots, L-1\}$ ab, und zwar so, daß für alle $l \in \{0, \ldots, L-1\}$ gilt $\iota^{-1}(\iota(l)) = l$. (Für $l' \notin \iota(\{0, \ldots, L-1\})$ ist $\iota^{-1}(l')$ beliebig.) Identifiziert man $F = \mathrm{GF}(2^{u_k})$ mit $\{0,1\}^{u_k}$, so kann man einfach jedem l seine Binärdarstellung (mit führenden Nullen) zuordnen.

Offensichtlich muß gelten $L \leq |F|$. Wegen Abschnitt 2.1.3.3 wird genügen $L \leq 2^{\tau}$.

Neben den r_i werden im Vorbereitungsprotokoll noch zufällig gewählte Werte $s_i \in F$ mit Protokoll 2.5 aufgeteilt. Erhält P_i sein Geheimnis s_i von einem anderen Prozessor, so überzeugt dies P_i davon, daß wenigstens $t+1$ Prozessoren sich vorläufig für den Wert „1“ entschieden haben (s_i ist also eine Art Paßwort gegenüber P_i).

Im eigentlichen Verteilungsprotokoll steht *kein* zuverlässiger Verteilkanal zur Verfügung. **Verteilt** ein Prozessor eine Nachricht N, so ist damit nur gemeint, daß er N an alle anderen Prozessoren über sichere Verbindungen sendet.

Protokoll 2.7: Byzantinische Verteilung mit linear kleinem Fehler mit Parametern L und τ

Passend zu n und τ werden gemäß Abschnitt 2.1.3.3 die Parameter der Protokolle 2.5 und 2.6 so gewählt, daß ihre Fehlerwahrscheinlichkeit jeweils $\leq 2^{-\tau}$ ist. F sei der gemäß Abschnitt 2.1.3.3 gewählte endliche Körper.

Phase 1: Vorbereitungsprotokoll

Jeder Prozessor P_i wählt zufällig Werte $r_i \in \{0, \ldots, L-1\}$ und $s_i \in F$. Die Werte $r_i^* := \iota(r_i)$ werden mittels Phase 1 von Protokoll 2.6 aufgeteilt, die Werte s_i mittels Phase 1 von Protokoll 2.5.

Phase 2: Verteilungsprotokoll

[0,a] Der Transmitter T verteilt seinen Anfangswert $v \in \{0, 1\}$.

Sei v_i der Wert, den Prozessor P_i vom Transmitter erhält. Jeder Prozessor P_i initialisiert einen leeren Vektor $E_i = (E_{i,1}, \ldots, E_{i,n}) \in (F^n)^n$.

[0,b] Ist $v_i = 1$, so führt Prozessor P_i für alle Geheimnisse s_j Phase 2, Schritt [1] von Protokoll 2.5 aus (d. h. er verteilt seine Anteile an s_j zusammen mit den passenden Authentikationsnachweisen).

Für $l = 1, \ldots, L-1$:

[l,a] Wenn ein Prozessor P_i in [l–1,b] Anteile von einem anderen Prozessor P_k erhält, so prüft er, ob er sie akzeptieren kann (gemäß Phase 2, Schritt [2] von Protokoll 2.5). Wenn nicht, so akzeptiert er keinen der Anteile von P_k.

Akzeptiert P_i die Anteile von P_k, so nimmt er sie in den Vektor E_i an die k-te Stelle auf (d. h. $E_{i,k}$ enthält alle Anteile von P_k an allen s_j). P_i nimmt seine eigenen Anteile in E_i genau dann auf, wenn er sie verteilt hat.

Wenn nun E_i erstmals wenigstens $t+1$ Einträge enthält, so verwendet P_i für jedes j Phase 2, Schritt [3] von Protokoll 2.5, um s_j zusammenzusetzen. Gelingt dies, so sendet er s_j an P_j (ansonsten disqualifiziert er lokal P_j).

[l,b] Erhält P_i in [l,a] von einem Prozessor P_k erstmals einen Wert x_i, der sein Geheimnis s_i sein soll, so prüft er diesen.

Ist $x_i = s_i$ und hat P_i sein Geheimnis s_i in Schritt [l,a] erstmals erhalten und hat seine Anteile in Schritt [0,b] noch nicht verteilt, so verteilt er sie nun. Er geht hierfür wie in Schritt [0,b] vor.

Ist $x_i \neq s_i$, so wird P_i von P_k keinen Wert x_i mehr betrachten.

[L,a] Wenn ein Prozessor P_i in [L–1,b] Anteile von einem anderen Prozessor P_k erhält, so prüft er, ob er sie akzeptieren kann. Wenn nicht, so akzeptiert er keinen der Anteile von P_k.

Akzeptiert P_i die Anteile von P_k, so nimmt er sie in den Vektor E_i an die k-te Stelle auf. P_i nimmt seine eigenen Anteile in E_i genau dann auf, wenn er sie in [L–1,b] verteilt hat.

Für jedes $l \in \{0, \ldots, L-1\}$ und jeden Prozessor P_i sei
$v_i^{(l)} := 1$, falls E_i nach Schritt [l+1,a] wenigstens $t+1$ Einträge enthielt, und
$v_i^{(l)} := 0$ sonst.

[L+1] Jeder Prozessor P_i nimmt am Zusammensetzen aller Geheimnisse r_j^* teil (mittels Phase 2 von Protokoll 2.6) und berechnet die Summe R aller $\iota^{-1}(r_j^*)$ modulo L.

Der Endwert von P_i ist $v_i^{(R)}$.

Lemma 2.7: (Münzwurf)

Ist die Fehlerwahrscheinlichkeit von Protokoll 2.6 $\leq 2^{-\tau}$ und ist $t < n/2$, so gilt:

a) Mit Wahrscheinlichkeit $\geq 1 - n2^{-\tau}$ gelten folgende beiden Aussagen:
   a1) Nach Phase 2, Schritt [L+1] erhalten alle korrekten Prozessoren denselben Wert für R.
   a2) Vor Phase 2, Schritt [L+1] haben die Störer keine Information über R.

b) Der Wert R ist in $\{0, \ldots, L-1\}$ fast gleichverteilt, d. h. für alle $l \in \{0, \ldots, L-1\}$ gilt $\mid W(R = l) - L^{-1} \mid \leq n2^{-\tau}$. ♦

Der Teil des Protokolls zur Bestimmung von R wird üblicherweise **Münzwurfprotokoll** genannt, R selbst das Ergebnis eines **fairen Münzwurfs**.

Lemma 2.7 wurde getrennt von Satz 2.8 formuliert, da ein Münzwurfprotokoll auch zum Kern vieler anderer Verteilungsprotokolle gehört (z. B. [Rabi_83]) und daher auch als eigenständiges Protokoll von Interesse ist.

Satz 2.8: (Byzantinische Verteilung mit linear kleinem Fehler)

Seien die Fehlerwahrscheinlichkeiten der Protokolle 2.5 und 2.6 jeweils $\leq 2^{-\tau}$ und ihre Parameter entsprechend Abschnitt 2.1.3.3 gewählt. Ist $t < n/2$, so gilt:

a) Gelten für alle $2n$ Ausführungen von Protokoll 2.5 und Protokoll 2.6 die Aussagen von Satz 2.5b bzw. Satz 2.6b fehlerfrei, so gilt:
   a1) Erhalten in Phase 2, Schritt [0,a] alle korrekten Prozessoren denselben Wert $v_i = 1$, so akzeptieren sie alle diesen Wert.
   a2) Erhalten in Phase 2, Schritt [0,a] alle korrekten Prozessoren denselben Wert $v_i = 0$, so akzeptieren sie alle diesen Wert mit Wahrscheinlichkeit $\geq 1-2^{-\tau}$.
   a3) Für alle $l \in \{0, \ldots, L-1\}$ und korrekten Prozessoren P_i und P_j gilt:
   Ist $l > 0$ und $v_i^{(l-1)} = 1$, so ist $v_j^{(l)} = 1$.
   Ist $l < L-1$ und $v_i^{(l+1)} = 0$, so ist $v_j^{(l)} = 0$.
   a4) Erhalten in Phase 2, Schritt [0,a] die korrekten Prozessoren verschiedene Werte, so akzeptieren sie mit Wahrscheinlichkeit $\geq 1-L^{-1}$ dennoch alle denselben Wert.

   Die Bedingung von (a) trifft mit Wahrscheinlichkeit $\geq 1 - 2n2^{-\tau}$ zu.

b) Der Rechenaufwand des Protokolls ist polynomial in n, L und τ.

   Das Vorbereitungsprotokoll benötigt im schlimmsten Fall $10t+17$ Runden, in denen $O(\tau\,(\tau+\log(n))^2\, n^6)$ bit zuverlässig verteilt und über sichere nichtabhörbare Verbindungen gesendet werden.

   Das Verteilungsprotokoll benötigt stets $2L+1$ Runden, in denen im schlimmsten Fall in höchstens $3n^2$ Nachrichten $O((\tau+\log(n))^2 n^4)$ bit über sichere, nicht notwendigerweise nichtabhörbare Verbindungen gesendet werden. ♦

Ist der Transmitter korrekt, so hat das Protokoll eine exponentiell kleine Fehlerwahrscheinlichkeit (Satz 2.8a1,2), ist er fehlerhaft, eine Fehlerwahrscheinlichkeit exponentiell nahe $1/L$ (Satz 2.8a4). Der zusätzliche exponentiell kleine Fehler in Satz 2.8a2 entsteht durch die exponentiell kleine Chance der Störer, in den Schritten [l,b] Geheimnisse s_i korrekter Prozessoren zu erraten.

Satz 2.8a3 wird in Abschnitt 2.2.3 benötigt.

Beweis von Lemma 2.7:

a) Es wird angenommen, für alle n Ausführungen von Protokoll 2.6 gelten die Aussagen von Satz 2.6b fehlerfrei. Diese Annahme trifft mit Wahrscheinlichkeit $\geq 1 - n2^{-\tau}$ zu.

a1) Nach Satz 2.6b1 ist kein korrekter Prozessor disqualifiziert, und nach Satz 2.6b2 erhalten daher alle dasselbe R.

a2) Ist P_i korrekt und nicht disqualifiziert, so erhalten nach Satz 2.6a die Störer vor Schritt [L+1] keine Information über r_i. Insbesondere müssen sie ihre eigenen r_j, die in [L+1] zusammengesetzt werden, zuvor unabhängig von r_i wählen (Satz 2.6b2). Da r_i zufällig aus {0, …, L–1} gewählt ist, ist $R = r_1 + \ldots + r_i + \ldots + r_n$ in {0, …, L–1} gleichverteilt. Nach Satz 2.6b1 wird P_i nicht disqualifiziert.

b) Trifft die obige Annahme, A abgekürzt, zu, so ist R in {0, …, L–1} gleichverteilt. Aus $W(A) \geq 1 - n2^{-\tau}$ folgt dann für jedes l

$$L^{-1}-n2^{-\tau} \leq L^{-1}(1-n2^{-\tau}) \leq W(A)\, L^{-1} \leq W(R = l) \leq W(A)\, L^{-1} + W(\neg A) \leq L^{-1}+n2^{-\tau}$$

d. h. $|W(R = l) - L^{-1}| \leq n2^{-\tau}$. ❐

Da jede Menge von t+1 Prozessoren wenigstens einen korrekten enthält, würde es in Protokoll 2.7 auch genügen, wenn irgendwelche t+1 Prozessoren Zufallswerte r_i wählen und im Vorbereitungsprotokoll aufteilen [BrDo_84]. Die Verwendung eines Protokolls zur überprüften Geheimnisaufteilung zur Konstruktion einer fairen Münze wurde erstmals in [CGMA_85] beschrieben.

Beweis von Satz 2.8:

a1) Angenommen, nach Schritt [0,a] haben alle korrekten Prozessoren $v_i = 1$.

Dann entscheiden alle korrekten Prozessoren in Schritt [0,b], ihre Anteile an allen Geheimnissen s_j zu verteilen. Nach Satz 2.5b1 werden alle korrekten Prozessoren ihre Anteile gegenseitig akzeptieren. Folglich haben nach diesem und allen folgenden Schritten alle korrekten Prozessoren wenigstens $t+1$ Einträge in ihren Vektoren. Daher entscheiden sich alle für „1“, unabhängig vom Resultat des Münzwurfs in Schritt [L+1].

a2) Angenommen, nach Schritt [0,a] haben alle korrekten Prozessoren $v_i = 0$.

Man muß zeigen, daß fast sicher kein korrekter Prozessor P_i seine Anteile verteilen wird. Dann hat kein korrekter Prozessor P_j einen Anteil eines anderen korrekten Prozessors in seinem Vektor E_j. Folglich hat nach jedem Schritt P_j

höchstens t Einträge, und P_j wird sich für „0“ entscheiden, unabhängig vom Resultat des Münzwurfs in Schritt [L+1]. (Dies kann allerdings auch unter der Bedingung von (a) nicht fehlerfrei gelten, denn erraten die Störer zufällig das Geheimnis s_i eines korrekten Prozessors P_i, so wird P_i fälschlich seine Anteile verteilen.)

Zum Beweis nehme man im Widerspruch zur obigen Behauptung an, es gäbe einen ersten Schritt [l], in dem ein korrekter Prozessor P_i seine Anteile verteilt. Es ist $l \neq 0$, da $v_i = 0$. Folglich hat P_i sein Geheimnis s_i von einem Prozessor P_j in Schritt [l,a], $l > 0$, erhalten. P_j muß fehlerhaft sein, denn kein korrekter Prozessor hatte bisher irgendeinen Anteil eines korrekten Prozessors in seinem Vektor. Da aber bislang kein korrekter Prozessor einen Anteil verteilt hat, können die Störer s_i nur erraten (Satz 2.5a1).

In allen L Runden kann jeder Störer an jeden korrekten Prozessor P_i höchstens ein „Geheimnis“ s_i senden; die Störer haben also weniger als n^2 Rateversuche. Nach jedem erfolglosen Rateversuch bei P_i können die Störer ein s_i ausschließen. Nach n Versuchen ist die Ratewahrscheinlichkeit je korrektem Prozessor daher $(|F|-n)^{-1}$, für alle korrekten Prozessoren zusammen daher

$$\leq \frac{n^2}{|F|-n} \leq 2n^2\ |F|^{-1} \leq 2^{-\tau},$$

wobei die beiden letzten Ungleichungen aus den Festlegungen von Abschnitt 2.1.3.3 folgen (mit u_k und k wie dort):

$$|F| = 2^{u_k} > 2^{k+1} \geq 2^{\tau + 5\log_2(n) + 3} = 8\, n^5\, 2^{\tau},$$

also auch $|F| \geq 2n$ (deshalb gilt die erste Ungleichung) und $|F| \geq n^2 2^{\tau+1}$.

a3) Ist $l > 0$ und $v_i^{(l-1)} = 1$, so enthält E_i nach Schritt [l,a] wenigstens t+1 Einträge, d. h. wenigstens t+1 Anteile an jedem Geheimnis, und P_i wird daher spätestens jetzt die Geheimnisse s_j der korrekten P_j korrekt zusammensetzen (Satz 2.5b2) und an die P_j senden. Jeder korrekte P_j wird daher in Schritt [l,b] seine Anteile verteilen, und alle korrekten Prozessoren werden diese Anteile akzeptieren (Satz 2.5b1) und nach Schritt [l+1,a] (und allen folgenden) wenigstens t+1 Einträge in ihren Vektoren haben, d. h. für alle korrekten Prozessoren P_j gilt $v_j^{(l)} = 1$.

Ist $l < L-1$ und $v_i^{(l+1)} = 0$, und würde für einen korrekten Prozessor P_j gelten $v_j^{(l)} = 1$, so würde nach dem eben gezeigten gelten $v_i^{(l+1)} = 1$, im Widerspruch zur Voraussetzung.

a4) Nach (a3) kann es nur einen Schritt [l], $l \in \{0, \ldots, L-1\}$, in Phase 2 geben, nach dem keine Konsistenz herrscht: Herrscht in [l] keine Konsistenz, so gibt es einen Prozessor P_i, für den $v_i^{(l)} = 1$ gilt, und nach (a3) wird dann nach allen folgenden Schritten $l' > l$ für alle P_j gelten $v_j^{(l')} = 1$.

Nach Lemma 2.7a2 (das unter der Bedingung von (a) fehlerfrei bewiesen wurde) ist R in $\{0, \ldots, L-1\}$ gleichverteilt und die Störer haben vor Schritt [L+1] keine Information über R. Das Verhalten der Störer bis Schritt [L+1] ist daher un-

abhängig von R. Die Wahrscheinlichkeit, daß am Ende keine Konsistenz herrscht, ist daher kleiner als die Wahrscheinlichkeit für $R = l$, und diese ist für jedes l gerade $1/L$.

b) Da alle Grundprotokolle einen in τ und n polynomialen Rechenaufwand haben, ist der erste Teil trivial (Lemma 2.3c, Lemma 2.4c, Satz 2.5c, Satz 2.6c und Abschnitt 2.1.3.3).

Das Vorbereitungsprotokoll besteht aus jeweils n parallelen Ausführungen der Phasen 1 von Protokoll 2.5 und Protokoll 2.6. Nach Lemma 2.3c und Lemma 2.4c erfordert dies insgesamt $10t+17$ Runden, in denen $O(mn^6k^2)$ bit veröffentlicht und über sichere nichtabhörbare Verbindungen gesendet werden. Nach Abschnitt 2.1.3.3 ist $m = O(\tau)$ und $k = O(\tau + \log(n))$, so daß gilt:

$$O(mn^6k^2) = O(\tau\,(\tau + \log(n))^2 n^6).$$

Das Verteilungsprotokoll benötigt für jeden Schritt außer Schritt [L,a] eine Runde (Satz 2.6c), also insgesamt $2L+1$ Runden.

In Schritt [0,a] sendet der Transmitter $O(n)$ bit in $n-1$ Nachrichten.

In Schritt [0,b] bis Schritt [$L-1$,b] sendet *jeder* Prozessor insgesamt höchstens einmal ein Geheimnis s_i an jeden Prozessor P_i, also $O(nk)$ in höchstens $n(n-1)$ Nachrichten, und höchstens einmal seine Anteile an allen s_j an alle, also $O(n^2k^2)$ bit in höchstens $n(n-1)$ Nachrichten.

In Schritt [$L+1$] wird n mal Phase 2 von Protokoll 2.6 ausgeführt. Dies benötigt nach Satz 2.6c $O(n^4k^2)$ bit = $O((\tau + \log(n))^2\, n^4)$ bit. Faßt man wie üblich alle Nachrichten von P_i an P_j in einer Runde zu einer zusammen, so muß jeder Prozessor an jeden anderen eine Nachricht senden, also insgesamt höchstens $n(n-1)$ Nachrichten.

Zusammen ergibt dies einen Nachrichtenaufwand von

$$n-1 + 3n(n-1) = 3n^2 - 2n - 1 < 3n^2$$

Nachrichten und $O((\tau + \log(n))^2\, n^4)$ bit. ❐

Die im Vorbereitungsprotokoll aufgeteilten Werte r_i, s_i können nur für *eine* Ausführung von Protokoll 2.7 verwendet werden. Soll Protokoll 2.7 mehrfach, etwa u mal, ausgeführt werden, so erhöht sich daher der Aufwand zur Vorbereitung gerade um den Faktor u. Dasselbe gilt auch für die folgenden beiden Verteilungsprotokolle.

Wünschen würde man sich, mit einem „kurzen“ Vorbereitungsprotokoll „viele“ Verteilungen durchführen zu können. Auf dieses Problem wird in Abschnitt 4.3 (aufbauend auf Abschnitt 3.1) nochmals zurückgekommen.

2.2.2 Byzantinische Verteilung mit Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ in $O(\sigma)$ Runden

Starten im Protokoll 2.7 alle korrekten Prozessoren mit demselben Wert $v_i = v$,

so enden fast sicher alle korrekten Prozessoren mit dem Endwert v. Das Protokoll „bewahrt“ also eine einmal gefundene korrekte und konsistente Lösung. Iteriert man es nun K mal, wobei als Startwerte v_i für eine Iteration jeweils die Endwerte der vorigen Iteration verwendet werden, so wird nur dann *keine* Lösung gefunden, wenn in *keiner* der K Iterationen eine gefunden wird. Die Wahrscheinlichkeit hierfür ist exponentiell nahe L^{-K}, also selbst exponentiell klein.

Dies führt zu folgendem Protokoll 2.8. Es benötigt ein Vorbereitungsprotokoll, zu dessen Ausführung ein zuverlässiger Verteilkanal und nichtabhörbare Verbindungen vorausgesetzt werden, und in dem K Verteilungen nach Protokoll 2.7 vorbereitet werden.

Mit Parameter σ hat das Protokoll eine Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$. L wird im folgenden eine kleine Konstante sein, z. B. $L = 2, 3$.

Protokoll 2.8: Byzantinische Verteilung mit Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ in $O(\sigma)$ Runden mit Parametern L und σ

Zu den Parametern L und σ werden bestimmt

$$K := \lceil \frac{\sigma+1}{\log_2(L)} \rceil$$

und

$$\tau := \sigma + \log_2(n) + \log_2(K) + \log_2(6).$$

Passend zu n und τ werden gemäß Abschnitt 2.1.3.3 die Parameter der Protokolle 2.5 und 2.6 so gewählt, daß ihre Fehlerwahrscheinlichkeit jeweils $\leq 2^{-\tau}$ ist.

Phase 1: Vorbereitungsprotokoll

Jeder Prozessor P_i wählt zufällig $2K$ Werte $r_{i,u} \in \{0, \ldots, L-1\}$ und $s_{i,u} \in F$, jeweils $u = 1, \ldots, K$. Die Werte $r^*_{i,u} := \iota(r_{i,u})$ werden mittels Phase 1 von Protokoll 2.6 aufgeteilt, die Werte $s_{i,u}$ mittels Phase 1 von Protokoll 2.5.

Phase 2: Verteilungsprotokoll

[0] Der Transmitter T verteilt seinen Anfangswert $v \in \{0, 1\}$.

Sei $v_{i,0}$ der Wert, den P_i vom Transmitter erhält.

Für $u := 1, \ldots, K$:

[u] Phase 2 von Protokoll 2.7 mit Parametern L und τ wird ausgeführt, allerdings ohne Schritt [0,a]. Die Startwerte des Protokolls (dort v_i) sind statt dessen definiert durch

$$v_{i,u-1}.$$

Intern werden die Werte $r_{i,u}$, $s_{i,u}$ verwendet. Der Endwert von P_i aus Protokoll 2.7 sei $v_{i,u}$.

[K+1] Der Endwert von P_i ist $v_{i,K}$.

Satz 2.9: (Byzantinische Verteilung mit Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$)
Ist $t < n/2$ und $L \geq 2$, so gilt:

a) Ist $t < n/2$, so wird in jedem Protokollablauf mit Wahrscheinlichkeit $\geq 1-2^{-\sigma}$ das byzantinische Verteilungsproblem gelöst.

b) Der Rechenaufwand des Protokolls ist polynomial in n, L und σ.

Das Vorbereitungsprotokoll benötigt im schlimmsten Fall $10t+17$ Runden, in denen $O(\sigma(\sigma + \log(n))^3 n^6)$ bit zuverlässig verteilt und über sichere nichtabhörbare Verbindungen gesendet werden.

Das Verteilungsprotokoll benötigt stets $2KL+1$ Runden, in denen in höchstens $(3\sigma+6)n^2$ Nachrichten $O(\sigma(\sigma + \log(n))^2 n^4)$ bit über sichere, nicht notwendigerweise nichtabhörbare Verbindungen gesendet werden.

c) Ist $\sigma \geq 27$, so wird die Rundenanzahl für $L = 3$ minimal. Für natürliche $\sigma < 27$ sind $L = 2$ oder $L = 3$ optimal. ♦

In Protokoll 2.8 ist L nur noch ein Leistungsparameter, den man sinnvollerweise konstant wählt. Damit die Fehlerwahrscheinlichkeit $2^{-\sigma}$ in Satz 2.9 exponentiell klein ist, muß L polynomial in n und σ sein.

Beweis:

a) Vorbemerkung: Es werden nur die Fälle betrachtet, in denen die jeweils Kn Ausführungen von Protokoll 2.5 und Protokoll 2.6 fehlerfrei arbeiten, d.h. in denen für alle Kn Ausführungen von Protokoll 2.5 bzw. 2.6 die Aussagen von Satz 2.5b bzw. 2.6b fehlerfrei gelten. Unter dieser Bedingung gilt für alle K Iterationen von Protokoll 2.7 Satz 2.8a.

Weiter werden nur die Fälle betrachtet, in denen für alle K Iterationen von Protokoll 2.7 auch Satz 2.8a2 fehlerfrei gilt.

Insgesamt trifft die Vorbemerkung mit Wahrscheinlichkeit

$$\geq 1 - 3Kn2^{-\tau}$$

zu. (Für Satz 2.8a2 wird $2^{-\tau} \leq n2^{-\tau}$ verwendet.)

Notation: Im folgenden ende ein Schritt $[u]$ **konsistent** (mit Wert v), wenn für alle korrekten Prozessoren P_i, P_j gilt $v_{i,u} = v_{j,u}$ $(= v)$. Er ende **inkonsistent**, wenn dies nicht zutrifft.

Fall 1: Ist der Transmitter korrekt, so endet Schritt [0] konsistent mit einem Wert v. Nach Satz 2.8a1,2 enden dann auch alle Schritte $[u]$, $u = 1, \ldots, K$, konsistent mit demselben Wert v.

Fall 2: Sei der Transmitter fehlerhaft und ende Schritt [0] inkonsistent.

Dann endet mit Wahrscheinlichkeit $\leq L^{-1}$ Schritt [1] inkonsistent, und mit Wahrscheinlichkeit $\leq L^{-u}$ jeder Schritt $[u]$, $u = 1, \ldots, K$.

Der Fehler in Fall 2 und der der Vorbemerkung ergeben zusammen eine Fehlerwahrscheinlichkeit

$$\leq L^{-K} + 3Kn2^{-\tau}.$$

Beide Summanden sind mit den obigen Festlegungen für K und τ je $\leq 2^{-\sigma-1}$:

$$L^{-K} \leq 2^{-\sigma-1} \Leftrightarrow K \geq \frac{\sigma+1}{\log_2(L)}$$

und $$3Kn2^{-\tau} \leq 2^{-\sigma-1} \Leftrightarrow \tau \geq \sigma + \log_2(n) + \log_2(K) + \log_2(6).$$

b) Mit den gemachten Festlegungen für τ und K folgt aus $L \geq 2$:

$$K \leq \frac{\sigma+1}{\log_2(L)} + 1 \leq \sigma+2 = O(\sigma),$$

woraus auch $\tau = O(\sigma + \log(n))$ folgt. Aus Satz 2.8c folgt dann, daß der Rechenaufwand des Protokolls polynomial in n, L und σ ist.

Zum Vorbereitungsprotokoll: Dieses besteht aus $K = O(\sigma)$ parallelen Ausführungen von Protokoll 2.7, Phase 1. Der Rest folgt aus Satz 2.8b.

Zum Verteilungsprotokoll: Protokoll 2.8 besteht aus K Iterationen von Protokoll 2.7, wobei allerdings Schritt [0,a] von Protokoll 2.7, die Verteilung des Anfangswerts des Transmitters, nur einmal ausgeführt wird.

Die Rundenanzahl ist daher $2KL+1$, und nach Satz 2.8b werden in höchstens $3Kn^2 \leq (3\sigma+6)n^2$ Nachrichten $O(\sigma(\sigma+\log(n))^2n^4)$ bit versendet.

c) Aus der Festlegung für K ergibt sich als Rundenanzahl

$$2KL+1 = 2\,L\left\lceil\frac{\sigma+1}{\log_2(L)}\right\rceil.$$

Die Funktion $f(L) := L/\log_2(L)$ hat ihr einziges Minimum an der Stelle $L = e$ (EULERsche Konstante, = 2,71…), wie man durch Ableiten nach L leicht verifiziert. Als natürliche Annäherungen kommen $L = 2, 3$ in Frage. Nachrechnen zeigt $f(2) = 2$ und $f(3) = 1{,}89\ldots$, d. h. das natürliche Minimum ist an der Stelle $L = 3$.

Für hinreichend große σ ist $L = 3$ damit auch für die Rundenanzahl optimal:

Für $$6\left\lceil\frac{\sigma+1}{\log_2(3)}\right\rceil \le 2L\left\lceil\frac{\sigma+1}{\log_2(L)}\right\rceil$$

ist hinreichend

$$6\left(\frac{\sigma+1}{\log_2(3)}+1\right)\le 2L\left(\frac{\sigma+1}{\log_2(L)}\right)\Leftrightarrow \sigma\ge\left(\frac{3}{\frac{L}{\log_2(L)}-\frac{3}{\log_2(3)}}\right)-1.$$

Die rechte Seite wird am größten, wenn man einen Wert L wählt, für den $f(L)$ dem Wert $f(3)$ am nächsten kommt. Da $f(L)$ für $L < e$ monoton fällt und für $L > e$ monoton wächst, kommen nur $L = 2, 4$ in Frage. Beide liefern $f(L) = 2$, d. h. es ist hinreichend

$$\sigma\ge\left(\frac{3}{2-\frac{3}{\log_2(3)}}\right)-1 = 26{,}98\ldots \Leftarrow \sigma\ge 27.$$

Für natürliche $\sigma < 27$ sind jeweils $L = 2$ oder $L = 3$ optimal: Für jedes $L > 4$ ist, da $f(L)$ dann monoton wächst,

$$2L\left\lceil\frac{\sigma+1}{\log_2(L)}\right\rceil \ge 2L\left(\frac{\sigma+1}{\log_2(L)}\right) > 4(\sigma+1).$$

Für $L > 4$ ist die Rundenanzahl daher größer als für $L = 4$, und damit auch größer als für $L = 2$. Der Wert für $L = 2$ kann nur durch $L = 3$ unterboten werden. ❒

Angenommen, ein Fehler 2^{-100} sei akzeptabel, d. h. $\sigma = 100$. Nach Satz 2.9 wird für das optimale $L = 3$ in Protokoll 2.8 der Wert $K = 64$ benötigt. Das Protokoll benötigt dann 385 Runden.

2.2.3 Byzantinische Verteilung mit Fehlerwahrscheinlichkeit $\le 2^{-\sigma}$ in erwartet 19 Runden

Das in Kapitel 3 beschriebene Verteilungsprotokoll benötigt stets $t+1$ Runden, für $t = n-2$ (der letzte sinnvolle Fall) also $n-1$ Runden. Im obigen Beispiel benötigt Protokoll 2.8 also erst für $n > 386$ Prozessoren weniger Runden.

Im folgenden Protokoll 2.9 wird Protokoll 2.8 daher so erweitert, daß es in einer erwartet konstanten Rundenanzahl terminiert (bei der günstigsten Wahl der Parameter: in weniger als 19). Im schlimmsten Fall benötigt es ebenso viele Runden wie Protokoll 2.8.

Die Grundidee ist, daß in Protokoll 2.7 ein korrekter Prozessor sich in manchen Situationen fast sicher davon überzeugen kann, daß das byzantinische Verteilungsproblem gelöst wurde:

Gilt am Ende des Protokolls

$$R \ne 0 \text{ und } v_i^{(R-1)} = 1,$$

so haben nach Satz 2.8a3 fast sicher *alle* korrekten Prozessoren P_j sich für $v_j^{(R)} = 1$

entschieden. Gilt

$$R \neq L\text{–}1 \text{ und } v_i^{(R+1)} = 0,$$

so haben nach Satz 2.8a3 fast sicher *alle* korrekten Prozessoren P_j sich ebenfalls für $v_j^{(R)} = 0$ entschieden.

Übertragen auf die Schritte von Protokoll 2.8, kann in diesen Situationen ein Prozessor seinen Endwert *vorzeitig* bestimmen.[25]

Damit P_i auch terminieren kann, muß sicher sein, daß die anderen korrekten Prozessoren ihren Endwert auch ohne P_i bestimmen können.

Da sich P_i schon entschieden hat, weiß er, daß sich in allen folgenden Iterationen von Protokoll 2.7 alle korrekten Prozessoren unabhängig vom jeweiligen Münzwurf für den Wert v_i entscheiden werden. Folglich kommt es in den folgenden Iterationen auf die *Unvorhersagbarkeit* der Münze nicht mehr an, und P_i kann seine Anteile an den Zufallswerten aller Prozessoren für *alle* Iterationen *sofort* nach seiner Entscheidung verteilen. Die Zufälligkeit der Münzwürfe bleibt dabei erhalten. Ist $v_i = 1$, so würde P_i zudem in allen Iterationen zu Beginn seine Anteile an allen Geheimnissen der anderen Prozessoren verteilen; da diese für die früheren Iterationen bedeutungslos sind, kann er auch dies *sofort* für *alle* Iterationen tun. Ist $v_i = 0$, so würde P_i in keiner der folgenden Iterationen seine Anteile verteilen.

Folglich muß P_i nach seiner Entscheidung nur noch in einer weiteren Runde *alle* Information verteilen, die er in Protokoll 2.8 verteilen würde, und kann dann terminieren.

Protokoll 2.9: Byzantinische Verteilung mit Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ in erwartet O(1) Runden mit Parametern L und σ

Zu den Parametern L und σ werden bestimmt

$$K := \left\lceil \frac{\sigma+1}{\log_2(L)} \right\rceil$$

und

$$\tau := \sigma + \log_2(n) + \log_2(K) + \log_2(6).$$

Passend zu n und τ werden gemäß Abschnitt 2.1.3.3 die Parameter der Protokolle

---

[25] Anhand der Eigenschaft, einen Endwert vorzeitig bestimmen zu können, wird das Byzantinische Verteilungsproblem üblicherweise in zwei Probleme aufgespalten: Für *Eventual Byzantine Agreement* (*EBA*) genügt es, wenn alle Prozessoren ihren Endwert irgendwann akzeptieren, für *Simultaneous Byzantine Agreement* (*SBA*) müssen sie ihn gleichzeitig akzeptieren [DoRS_90]. In diesem Sinne ist Protokoll 2.9 eine Lösung des EBA-Problems, alle anderen Protokolle in Kapitel 2 (und auch in Kapitel 3) Lösungen des SBA-Problems.

2.5 und 2.6 so gewählt, daß ihre Fehlerwahrscheinlichkeit jeweils $\leq 2^{-\tau}$ ist.

Phase 1: Vorbereitungsprotokoll

Jeder Prozessor P_i wählt zufällig $2K$ Werte $r_{i,u} \in \{0, \ldots, L-1\}$ und $s_{i,u} \in F$, jeweils $u = 1, \ldots, K$. Die Werte $r^*_{i,u} := \iota(r_{i,u})$ werden mittels Phase 1 von Protokoll 2.6 aufgeteilt, die Werte $s_{i,u}$ mittels Phase 1 von Protokoll 2.5.

Phase 2: Verteilungsprotokoll

[0] Der Transmitter T verteilt seinen Anfangswert $v \in \{0, 1\}$.

Sei $v_{i,0}$ der Wert, den P_i vom Transmitter erhält. P_i initialisiert $w_i := 0$.

Für $u := 1, \ldots, K$:

[u] Ist $w_i = 1$, so nimmt Prozessor P_i für alle $u' = u, \ldots, K$ teil an Phase 2, Schritt [1] von Protokoll 2.6 zum Zusammensetzen der Zufallswerte $r^*_{j,u'}$ (d. h. für jedes $r^*_{j,u'}$ verteilt er wie in Phase 2, Schritt [1] von Protokoll 2.5 alle seine Unteranteile an den Anteilen aller Prozessoren an $r^*_{j,u'}$, jeweils nebst passenden Authentikationsnachweisen; hierzu muß kein anderer Prozessor außer P_i irgendwelche Nachrichten senden). Ist $v_i = 1$, so verteilt P_i zusätzlich auch noch alle seine Anteile an den Geheimnissen $s_{j,u'}$ mit Phase 2, Schritt [1] von Protokoll 2.5. Danach **terminiert** P_i sofort.

Ist $w_i = 0$, so nimmt P_i teil an der Ausführung von Phase 2 von Protokoll 2.7 mit Parametern L und σ, allerdings ohne Schritt [0,a]. Sein Startwert für das Protokoll (dort v_i) ist statt dessen definiert durch

$$v_{i,u-1}.$$

Intern werden die Werte $r_{i,u}$, $s_{i,u}$ verwendet. Hat P_i die Anteile von P_k an den $s_{j,u}$ schon früher erhalten, so verwendet er diese so, als ob er sie in [0,b] von P_k erhalten hätte. Hat P_i die Information von P_k zum Zusammensetzen der $r_{j,u}$ schon früher erhalten, so verwendet er diese in [L+1]. Das Münzwurfergebnis sei jeweils R_u, die Zwischenergebnisse seien $v_{i,u}^{(l)}$, $l = 0, \ldots, L-1$, der Endwert sei $v_{i,u} := v_{i,u}^{(R_u)}$.

- Ist $u = K$, so entscheidet sich P_i für $v_i := v_{i,K}$ und **terminiert**.
- Ist $u < K$ und

  $$R_u \neq 0 \text{ und } v_{i,u}^{(R_u-1)} = 1$$

  oder

  $$R_u \neq L-1 \text{ und } v_{i,u}^{(R_u+1)} = 0,$$

  so sei $v_i := v_{i,u}$ und $w_i := 1$.

[K+1] Der Endwert von P_i ist v_i.

Satz 2.10: (Byzant. Verteilung mit Fehler $\le 2^{-\sigma}$ in erwartet O(1) Runden)
Ist $t < n/2$, so gilt:

a) In jedem Protokollablauf wird mit Wahrscheinlichkeit $\ge 1-2^{-\sigma}$ das byzantinische Verteilungsproblem gelöst.

b) Das Protokoll verursacht im schlimmsten Fall denselben Aufwand wie Protokoll 2.8.

Zu gegebenem Sicherheitsparameter σ wird die erwartete Rundenanzahl des Verteilungsprotokolls für $L = 2$ minimal. Ist $\sigma \ge 5$, so werden erwartet höchstens 19 Runden benötigt, in denen in höchstens $14n^2$ Nachrichten $O(\sigma(\sigma+\log(n))^2 n^4)$ bit über sichere, nicht notwendigerweise nichtabhörbare Verbindungen gesendet werden. ♦

Beweis:

a) Zunächst sieht man leicht, daß das Protokoll effektiv ausführbar ist: Ein Prozessor, der sich entschieden hat, kann tatsächlich alle geforderte Information vorzeitig verteilen, denn die Schritte [1] der Phasen 2 von Protokoll 2.5 und Protokoll 2.6 sind nicht interaktiv. Beim Zusammensetzen des Münzwurfs R_u können alle in Schritt $[u]$ noch nicht terminierten Prozessoren den Rest von Phase 2 von Protokoll 2.6 dann auch ohne die schon terminierten ausführen.

Zum Beweis der Korrektheit werden wie im Beweis von Satz 2.9a nur die Fälle betrachtet, in denen die jeweils Kn Ausführungen von Protokoll 2.5 und Protokoll 2.6 fehlerfrei arbeiten, und in denen für alle bis zu K Iterationen von Protokoll 2.7 Satz 2.8a2 fehlerfrei zutrifft.

Gezeigt wird, daß Protokoll 2.9 dann dasselbe Ergebnis liefert wie Protokoll 2.8 (mit derselben Phase 1 (Vorbereitungsprotokoll) und demselben Schritt [0] in Phase 2). Da im folgenden kein neuer Fehler hinzukommt, bleibt er $\le 2^{-\sigma}$.

Setzt *kein* korrekter Prozessor seinen Endwert vorzeitig, so liefern Protokoll 2.8 und Protokoll 2.9 offensichtlich identische Ergebnisse.

Sonst sei P_i der erste korrekte Prozessor, der in einem Schritt $[u]$, $u < K$, sein v_i vorzeitig bestimmt. Dann gilt

$$(R_u \ne 0 \text{ und } v_{i,u}{}^{(R_u-1)} = 1) \text{ oder } (R_u \ne L-1 \text{ und } v_{i,u}{}^{(R_u+1)} = 0).$$

Nach Satz 2.8a3 gilt dann für *jeden* korrekten Prozessor P_j

$$v_{j,u}{}^{(R_u)} = 1 \text{ bzw. } v_{j,u}{}^{(R_u)} = 0,$$

d. h. in Schritt $[u]$ wird das konsistente Ergebnis $v = v_i$ gefunden.

In Protokoll 2.8 würden sich daher nach Satz 2.8a1,2 alle korrekten Prozessoren P_j in allen folgenden Schritten $[u+h]$, $h = 1, 2, \ldots$, unabhängig vom jeweiligen Münzwurf R_{u+h} für $v_{j,u+h} = v$ entscheiden.

Dasselbe ist auch hier der Fall: Man betrachte zunächst Schritt $[u+1]$. Dieser beginnt konsistent mit $v_{j,u} = v$ für alle P_j. Ist $v = 0$, so haben die schon ent-

schiedenen Prozessoren ihre Anteile an den $s_{k,u+1}$ nicht verteilt, und da die konsistente Entscheidung nach Satz 2.8a2 bewahrt bleibt, wird dies in diesem Schritt auch keiner der noch nicht entschiedenen Prozessoren tun. Ist $v = 1$, so werden in [0,b] alle noch nicht entschiedenen Prozessoren ihre Anteile an den $s_{k,u+1}$ verteilen; die schon entschiedenen Prozessoren haben dies bereits getan. Wie im Beweis von Satz 2.8a1 sieht man, daß sich dann jeder korrekte Prozessor in Schritt $[u+1]$ unabhängig vom Münzwurf R_{u+1} wieder für $v_{j,u+1} = v$ entscheiden wird, d. h. genauso wie in Protokoll 2.8.

Dasselbe gilt nun induktiv für $[u+h]$, $h = 1, 2, \ldots$.

b) Der erste Teil gilt offensichtlich, denn Protokoll 2.9 ist in Protokoll 2.8 enthalten. In Protokoll 2.9 wird dieselbe Information gesendet wie in Protokoll 2.8, folglich ist der Nachrichtenaufwand gemessen in bit gleich groß.

Zur erwarteten Runden- und Nachrichtenanzahl:

Vorbemerkung: Es werden nur dieselben Fälle betrachtet wie im Beweis von (a). Mit Wahrscheinlichkeit $\leq 2^{-\sigma}$ treffen diese Fälle hier nicht zu. Ist L konstant, so ist im ungünstigsten Fall die Rundenanzahl $O(\sigma)$, d. h. der Fehler kann den Erwartungswert höchstens um $O(\sigma 2^{-\sigma}) = O(1)$ erhöhen, d. h. um eine Konstante. Im folgenden wird $L = 2$ optimal sein. Für $L = 2$ ist $K \leq \sigma+2$, d. h. die Rundenanzahl ist im schlimmsten Fall $4K + 1 = 4\sigma+9$, und es ist $(4\sigma+9)\, 2^{-\sigma} \leq 1 \Leftrightarrow \sigma \geq 5$.

Entscheidet sich in Schritt $[u]$ ein Prozessor, so sind die Münzwürfe R_{u+h} in Schritt $[u+h]$ nicht mehr unvorhersagbar. Sie sind allerdings (unter der obigen Voraussetzung) nach wie vor in $\{0, \ldots, L-1\}$ gleichverteilt.

Schritt [0] benötigt eine Runde.

Ein Schritt $[u]$, $u > 0$, benötigt $2L$ Runden. Ist E der Erwartungswert für die Anzahl der Schritte bis zur Entscheidung aller Prozessoren, so werden hierfür $2EL+1$ Runden benötigt. Dazu kommt noch eine Runde, in der der letzte Prozessor seine Anteile verteilt, insgesamt also $2EL+2$ Runden.

Es werden dann höchstens $3En^2+n^2$ Nachrichten versendet.

Zur Bestimmung von E werden drei Protokollzustände betrachtet: Zu Beginn eines Schrittes $[u]$ gilt in Zustand …

A) Es herrscht keine Konsistenz.
B) Es herrscht zwar Konsistenz, aber es haben sich noch nicht alle korrekten Prozessoren entschieden.
C) Alle korrekten Prozessoren haben sich entschieden.

E ist gerade der Erwartungswert für die Anzahl der Übergänge von (A) nach (C). Hierzu werden zunächst die Übergangswahrscheinlichkeiten $W(X \to Y)$ bestimmt.

A) Beginnen nicht alle korrekten Prozessoren einen Schritt $[u]$ mit demselben Wert $v_{i,u-1}$, so gibt es höchstens ein $Q_u \in \{0, \ldots, L-1\}$, so daß gilt (Bild 2-2):

- Für alle korrekten P_i und alle $l < Q_u$ ist $v_{i,u}^{(l)} = 0$.
- Es gibt korrekte P_i und P_j, so daß $v_{i,u}^{(Q_u)} = 0$ und $v_{j,u}^{(Q_u)} = 1$.
- Für alle korrekten P_i und alle $l > Q_u$ ist $v_{i,u}^{(l)} = 1$.

Dies folgt unmittelbar aus Satz 2.8a3. Im schlimmsten Fall können die Störer Q_u frei bestimmen. (Existiert Q_u nicht, so herrscht Konsistenz.)

Zur Bestimmung der möglichen Übergänge betrachte man die „Umgebung“ von Q_u (Bild 2-2):

Runde l	$v_{i,u}^{(l)}$	$v_{j,u}^{(l)}$	A→
0	0	0	C
	…	…	C
Q_u–1	0	0	B
Q_u	1	0	A
Q_u+1	1	1	B
	…	…	C
L–1	1	1	C

Bild 2-2 Rundenergebnisse von P_i und P_j in Schritt [u]. In Runde $l = Q_u$ herrscht keine Konsistenz. Rechts stehen jeweils die Zustände, die von Zustand A aus erreicht werden, wenn $R_u = l$ gewählt wird.

Da in (A) noch kein Prozessor terminiert ist, ist R_u unvorhersagbar, d. h. zu dem Zeitpunkt, zu dem die Störer Q_u bestimmen, ist $W(R_u = l) = 1/L$ für jedes $l \in \{0, \ldots, L-1\}$.

A→A: Ist $R_u = Q_u$, so bleibt der Zustand erhalten. Es ist $W(A \to A) = L^{-1}$.

A→B: Ist $R_u \in \{Q_u-1, Q_u+1\}$, so herrscht zwar Konsistenz, aber es gibt wenigstens einen korrekten Prozessor, der sich nicht entscheiden kann (Bild 2-2: P_i bzw. P_j).

A→C: Ist $R_u \notin \{Q_u-1, \ldots, Q_u+1\}$, so können sich alle Prozessoren entscheiden.

Aus Sicht des Störers sind jeweils die Übergänge zu den lexikographisch kleineren Zuständen günstiger. Für die in diesem Sinne optimalen Störerstrategien sind $L = 2$ und $L \geq 3$ getrennt zu betrachten:

Ist $L = 2$, so ist $Q_u = 1$ optimal: Es ist dann sicher $R_u \in \{Q_u-1, \ldots, Q_u+1\}$, d. h. es ist kein Übergang zu (C) möglich. Folglich ist $W(A \to B) = 1/2$.

Ist $L \geq 3$, so ist jedes Q_u mit $1 \leq Q_u \leq L-2$ optimal, da dann alle 3 Werte $\{Q_u-1, Q_u, Q_u+1\}$ möglich sind, in denen *kein* Übergang nach (C) stattfindet. Es ist dann

$$W(A \to A) = 1/L,\ W(A \to B) = 2/L,\ W(A \to C) = 1-3/L.$$

B) Schritt [u] beginnen alle korrekten Prozessoren mit demselben Wert v, aber es

haben sich noch nicht alle entschieden. In diesem Fall gilt für alle l und alle P_i $v_{i,u}^{(l)} = v$, und folglich entscheiden sich in $[u]$ entweder alle oder keiner der bisher unentschiedenen Prozessoren.

B→B: Die bisher unentschiedenen Prozessoren entscheiden sich auch jetzt nicht, wenn $R_u = 0$ bzw. $R_u = L–1$ ist (für $v = 1$ bzw. 0). Also ist W(B→B) = $1/L$.

B→C: Dies ist sonst der Fall, also W(B→C) = $1–1/L$.

Sei jeweils $\ddot{u}_X$, X = A, B, die erwartete Anzahl der Übergänge, bis von (X) aus (C) erreicht wird. Es ist dann $E = \ddot{u}_A$. (Ist der Transmitter korrekt, so ist $E = \ddot{u}_B$.)

$\ddot{u}_A$ und $\ddot{u}_B$ werden anhand der folgenden MARKOV-Diagramme (Bild 2-3) bestimmt.

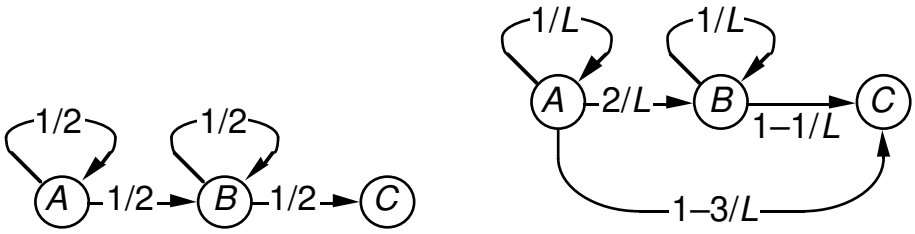


Bild 2-3 MARKOV-Diagramm für $L = 2$ MARKOV-Diagramm für $L \geq 3$

In beiden Diagrammen ist (C) von (B) aus gleich erreichbar, d. h. $\ddot{u}_B$ ist jeweils gleich definiert:

$$\ddot{u}_B = L^{-1}\,(1+\ddot{u}_B) + (1–L^{-1}) \Rightarrow \ddot{u}_B = L\,(L–1)^{-1}$$

Für $L = 2$ ist $\ddot{u}_B = 2$ und damit

$$\ddot{u}_A = 2^{-1}\,(1+\ddot{u}_A) + 2^{-1}\,(1+\ddot{u}_B) \Rightarrow \ddot{u}_A = 4.$$

Für $L \geq 3$ ist

$$\ddot{u}_A = L^{-1}\,(1+\ddot{u}_A) + 2L^{-1}\,(1+\ddot{u}_B) + 1 - 3L^{-1} \Rightarrow \ddot{u}_A = 2L\,(L–1)^{-2} + L(L–1)^{-1}.$$

Die Rundenanzahl $\rho(L)$ war (vgl. Seite 72) mit $E = \ddot{u}_A$ jeweils $\rho(L) := 2L\ddot{u}_A+2$, also

$$\rho(2) = 18,$$

und für $L \geq 3$: $\rho(L) = 4L^2\,(L–1)^{-2} + 2L^2(L–1)^{-1} + 2.$

Mit $L/(L–1) > 1$ kann man grob für $L \geq 3$ abschätzen

$$\rho(L) > 6+2L$$

und es ist $6+2L > 18 \Leftrightarrow L > 6$, d. h. für $L > 6$ ist die Rundenanzahl größer als für $L = 2$. Durch Nachrechnen findet man $\rho(3) = 20$, $\rho(4) = 19{,}7\ldots$, $\rho(5) = 20{,}75$, $\rho(6) = 22{,}16$.

Die minimale Rundenanzahl wird also für $L = 2$ erwartet, nämlich 18 Runden. Die Nachrichtenanzahl ist dann höchstens $13n^2$. Durch den in der Vorbe-

merkung gemachten Fehler erhöht sich die Rundenanzahl um 1 auf 19. Die Nachrichtenanzahl kann sich dann höchstens um n^2 erhöhen, denn mehr als n^2 Nachrichten können in einer Runde nicht gesendet werden. ❒

Protokoll 2.9 reduziert gegenüber Protokoll 2.8 zwar die erwartete Runden- und Nachrichtenanzahl, nicht aber den Nachrichtenaufwand in bit.

Auch dieser läßt sich im Erwartungswert auf den σ-ten Anteil des Aufwands von Protokoll 2.8 reduzieren, also auf $O((\sigma+\log(n))^2 n^4)$ bit. Da dies aber keinen wesentlichen Fortschritt darstellt, wird auf eine Ausformulierung der notwendigen Modifikationen von Protokoll 2.7 und Protokoll 2.9 hier verzichtet. Statt dessen wird nur die Idee skizziert:

Zur Motivation betrachte man Protokoll 2.9: Hat sich P_i in Schritt $[u]$ vorzeitig entschieden (also $R_u \neq 0$ und $v_{i,u}{}^{(R_u-1)} = 1$, oder $R_u \neq L-1$ und $v_{i,u}{}^{(R_u+1)} = 0$), so herrscht Konsistenz. Folglich entscheiden sich nach derselben Regel *alle* korrekten Prozessoren vorzeitig spätestens im nächsten Schritt $[v]$ mit $R_v \notin \{0, L-1\}$. Schließt man $R_v \in \{0, L-1\}$ für alle $v = 1, \ldots, K$ explizit aus (was $L \geq 4$ erzwingt), so ist $v = u+1$, d. h. P_i weiß, daß sich bereits in Schritt $[u+1]$ alle korrekten Prozessoren vorzeitig entscheiden werden. Folglich muß P_i nur noch an Schritt $[u+1]$ teilnehmen (und keine Information für die nachfolgenden Schritte $[u+2], \ldots, [K]$ verteilen). Da dasselbe für alle korrekten Prozessoren gilt, terminieren spätestens nach Schritt $[u+2]$ alle korrekten Prozessoren.

Zur Realisierung dieser Idee müssen die Protokolle 2.7 und 2.9 verändert werden: In Protokoll 2.7 wird R gleichverteilt aus $\{1, \ldots, L-2\}$ gezogen. Hierfür werden die $r_{i,u} \in \{0, \ldots, L-3\}$ gewählt und am Ende $R_u := 1 + (r_{1,u}+\ldots+r_{n,u} \bmod (L-3))$ berechnet. Dies beeinflußt nur die Fehlerwahrscheinlichkeiten in Lemma 2.7 und Satz 2.8; dort muß entsprechend L durch $L-2$ ersetzt werden. In Protokoll 2.9 entfällt in Schritt $[u]$ die nachrichtenaufwendige vorzeitige Verteilung der Anteile für alle späteren Schritte. Entscheidet sich P_i in Schritt $[u]$, $u < K$, so nimmt P_i noch an Schritt $[u+1]$ teil und terminiert dann. Um eine Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ zu erhalten, muß $K := \lceil(\sigma+1)/\log_2(L-2)\rceil$ gewählt werden.

Die erwartete Rundenanzahl $\rho(L)$ kann wie für Protokoll 2.9 bestimmt werden, wobei sich $L = 4$ als optimal erweist. Für $L = 4$ gilt: Schritt $[u]$ findet mit Wahrscheinlichkeit $\geq 1/2$ eine konsistente Lösung, also nach erwartet ≤ 2 Iterationen. Herrscht Konsistenz, so entscheiden sich *alle* Prozessoren spätestens nach der nächsten Iteration und terminieren spätestens nach der übernächsten, d. h. Schritt $[u]$ wird erwartet höchstens $E = 4$ mal iteriert. Insgesamt benötigt das Protokoll also erwartet höchstens $2LE + 2 = 34$ Runden. Der Summand 2 entstammt dabei Schritt [0] und der Kompensierung von Fehlerwahrscheinlichkeiten (vgl. Beweis von Satz 2.10)

2.3 Zusammenfassung

In diesem Kapitel wurden die ersten byzantinischen Verteilungsprotokolle beschrieben, die im byzantinischen Fehlermodell jedes $t < n/2$ tolerieren können.

Grundlage war ein Verfahren zur überprüften Geheimnisaufteilung (*verifiable secret sharing*) für $t < n/2$ mit exponentiell kleiner Fehlerwahrscheinlichkeit, das nur zum Aufteilen, nicht aber zum Zusammensetzen von Geheimnissen einen zuverlässigen Verteilkanal benötigt (Abschnitt 2.1).

Das einzige bislang bekannte Verfahren zur überprüften Geheimnisaufteilung für $t < n/2$, das von RABIN [RaBe_89], benötigt für beides einen zuverlässigen Verteilkanal. Es wurde so modifiziert (und verbessert), daß es der obigen Anforderung genügt.

Aufbauend auf dem Verfahren zur überprüften Geheimnisaufteilung wurde ein byzantinisches Verteilungsprotokoll entwickelt, das in $O(L)$ Runden eine Fehlerwahrscheinlichkeit $\approx L^{-1}$ erreicht (Abschnitt 2.2.1).

Durch Iteration wurde die Fehlerwahrscheinlichkeit in $O(\sigma)$ Runden kleiner als $2^{-\sigma}$ (Abschnitt 2.2.2). Hierfür müssen die korrekten Prozessoren in $O(\sigma n^2)$ Nachrichten $O(\sigma(\sigma + \log(n))^2 n^4)$ bit über sichere, aber nicht notwendigerweise nichtabhörbare Verbindungen, senden.

Durch Einführen eines Abbruchkriteriums wurde schließlich ein Protokoll erreicht, das mit derselben Fehlerwahrscheinlichkeit in erwartet höchstens 19 Runden terminiert. Im erwarteten Fall müssen die n Prozessoren in $O(n^2)$ Nachrichten $O(\sigma(\sigma + \log(n))^2 n^4)$ bit über sichere, aber nicht notwendigerweise nichtabhörbare Verbindungen, senden. Im schlimmsten Fall benötigt es ebensoviele Runden wie das Protokoll aus Abschnitt 2.2.2.

Im Vorbereitungsprotokoll der beiden Verteilungsprotokolle mit Fehler $\leq 2^{-\sigma}$ werden $O(\sigma n)$ Geheimnisse überprüft aufgeteilt (bzw. festgelegt). Hierfür werden $O(t)$ Runden benötigt, in denen im schlimmsten Fall $O(\sigma(\sigma + \log(n))^3 n^6)$ bit zuverlässig verteilt und über sichere nichtabhörbare Verbindungen gesendet werden müssen.

Für u Verteilungen erhöht sich der Aufwand der Verteilungsprotokolle (einschließlich der Vorbereitung) linear in u. Auf dieses Problem wird in Kapitel 4 näher eingegangen.

3 Byzantinische Verteilung mit beliebig vielen Störern in linearer Rundenanzahl

Im folgenden wird ein Protokoll zur byzantinischen Verteilung beschrieben, das im byzantinischen Fehlermodell jedes $t < n$ in $t+1$ Runden mit exponentiell kleiner Fehlerwahrscheinlichkeit toleriert.

Das Protokoll kann damit noch mehr Störer tolerieren als die aus Kapitel 2, benötigt dafür aber eine höhere Rundenanzahl. Auch dieses Protokoll ist das erste seiner Art.

Wesentlichstes Hilfsmittel ist ein Authentikationsmechanismus, der in vielen Protokollen, insbesondere in einem authentisierten Verteilungsprotokoll, die üblichen, nur kryptographisch sicheren digitalen Signaturen im byzantinischen Fehlermodell ersetzten kann (Abschnitt 3.1). Diese *Pseudosignaturen* sind eine Verallgemeinerung eines von CHAUM und ROIJAKKERS in [ChRo_91] vorgeschlagenen Verfahrens.

Das eigentliche Verteilungsprotokoll basiert auf einem authentisierten Verteilungsprotokoll von DOLEV und STRONG [DoSt_83: Th. 3] (Abschnitt 3.2).[26]

3.1 Pseudosignaturen

In Abschnitt 1.1.2.2 wurden die Eigenschaften eines für authentisierte Verteilungsprotokolle geeigneten sicheren Authentikationsmechanismus beschrieben.

Pseudosignaturen erfüllen die Eigenschaften der *Korrektheit* und *Unfälschbarkeit*. Die *Weitergebbarkeit* wird im folgenden Sinne garantiert: Für einen vorgegebenen Parameter λ kann eine *pseudosignierte*, d. h. authentisierte Nachricht N vom *Signierer* S aus wenigstens λ mal sequentiell weitergegeben werden, etwa entlang der Folge

$$S = P_{i_0} \to P_{i_1} \to \ldots \to P_{i_\lambda} \quad ,$$

so daß jeder j-te Empfänger P_{i_j} fast sicher ist, daß wenn er N akzeptiert, N auch vom $(j+1)$-ten Empfänger $P_{i_{j+1}}$ akzeptiert wird. P_{i_λ} kann die Nachricht nicht notwendigerweise ein $(\lambda+1)$-tes mal weitergeben.

In Abschnitt 1.1.2.2 wurde die Anzahl λ der Weitergaben nicht beschränkt. Die obige *λ-Weitergebbarkeit* genügt aber zur Implementierung eines authentisierten Verteilungsprotokolls in Abschnitt 3.2, und generell genügt sie für jedes

---

[26] Einen Überblick über die Ergebnisse dieses Kapitels enthält auch [PfWa_92].

authentisierte Protokoll, in dem die Anzahl der Weitergaben einer Nachricht polynomial begrenzt ist.

Der einzige bislang bekannte, für authentisierte Protokolle und $t \geq n/3$ und $\lambda > 2$ geeignete Authentikationsmechanismus waren die höchstens im kryptographischen Fehlfunktionsmodell sicheren digitalen Signaturen (Abschnitt 1.1.2.2.2). Im Gegensatz zu diesen beruhen Pseudosignaturen weder auf unbewiesenen Annahmen noch auf einem einschränkenden Fehlfunktionsmodell.

Das folgende Pseudosignaturschema ist eine Verallgemeinerung eines Verfahrens von CHAUM und ROIJAKKERS aus [ChRo_91], das speziell den Fall $\lambda = 2$ betrachtete.[27]

3.1.1 Grundidee

In digitalen Signatursystemen (Abschnitt 1.1.2.2.2) überprüfen alle Empfänger eine Signatur anhand desselben, allgemein bekannten Testschlüssels STe. Die wesentliche Grundidee für Pseudosignaturen ist, für *verschiedene* Empfänger *verschiedene* Testschlüssel zu verwenden [ChRo_91].

Der Testschlüssel STe_i, mit dem P_i mit SSi signierte Nachrichten testet, darf allerdings höchstens dem Signierer S und P_i bekannt sein; sonst könnte wieder eine Signatur mittels Durchprobieren aller möglichen Signaturen gefälscht werden. Der Fall, daß P_i eine Pseudosignatur akzeptiert und P_j nicht, ist daher nicht völlig auszuschließen (sonst wären $Test_{STe_i}$ und $Test_{STe_j}$ identisch!). Kennt S aber alle Testschlüssel STe_i, so ist die Konstruktion eines solch häßlichen Falles für S im Prinzip wieder möglich, d. h. S könnte P_i eine Pseudosignatur geben, die dieser nicht an P_j weitergeben kann. Also muß STe_i auch vor dem Signierer S verborgen sein.

Diese zunächst unmöglich klingende Anforderung wird in [ChRo_91] realisiert, indem die Testschlüssel dem Signierer S *anonym* zugesendet werden; d. h. S erfährt in einer Initialisierungsphase als einziger alle Testschlüssel, aber so, daß ihm verborgen bleibt, welcher zu welchem P_i gehört. Damit genügend viel geheime Information erzeugt wird, wird dies für ein vorgegebenes m, **Vielfachheit** genannt, m mal wiederholt: Die m Testschlüssel $(T_{i,1}, \ldots, T_{i,m})$ von

---

27 In [ChRo_91: §5] ist eine Erweiterung der dortigen Pseudosignaturen für $\lambda = 2$ skizziert, die die mehrfache Weitergebbarkeit ermöglichen sollte, für den allgemeinen Fall aber ungenügend ist: Sie setzt implizit voraus, daß der Signierer *korrekt* ist.

Ist der Signierer *nicht* korrekt, so kann es passieren, daß der zweite Empfänger akzeptiert und erwartet, daß auch der dritte akzeptiert, der dritte dies aber nicht tut. In [ChRo_91] ist dies zwar nicht erwähnt, aber leicht zu sehen: Die verwendeten Hash-Funktionen schützen nur gegen einen Störer, der sie nicht kennt. Sind, mit den dortigen Bezeichnungen, S und R_1 fehlerhaft, so kann deshalb ein R_2, der die Hälfte seiner *message numbers* und eine seiner (dann völlig bedeutungslosen) *checksums* findet, nicht sicher sein, daß auch R_3 die Hälfte seiner *message numbers* finden wird.

P_i bilden dann den Testschlüssel STe_i, die m Mengen $\{T_{1,j}, \ldots, T_{n,j}\}$, $j = 1, \ldots, m$, den Signierschlüssel SSi. (Dieses **Initialisierungsprotokoll** wird in Abschnitt 3.1.3 beschrieben).

Eine **Pseudosignatur** ψ besteht dementsprechend aus m **Abschnitten** ψ_j, und jeder Abschnitt aus n **Minisignaturen**, etwa $\psi_j = \{s_{1,j}, \ldots, s_{n,j}\}$, die zu den entsprechenden Testschlüsseln $\{T_{1,j}, \ldots, T_{n,j}\}$ passen. (Wie eine Minisignatur genau aussieht, wird in Abschnitt 3.1.2 beschrieben.)

Erhält P_i direkt vom Signierer S eine Pseudosignatur ψ unter eine Nachricht N, so kann er erwarten, daß in jedem Abschnitt ψ_j eine korrekte Minisignatur für ihn ist, d. h. daß ein $s \in \psi_j$ existiert mit $Test_{T_{i,j}}(N, s) =$ „ok“.

Würde man dies als einzige Akzeptionsbedingung verwenden, so könnte ein fehlerhafter S leicht erreichen, daß P_i eine akzeptierte Nachricht nicht mehr mit hinreichend großer Wahrscheinlichkeit weitergeben kann: Läßt S z. B. eine einzige Minisignatur aus ψ weg, so ist diese mit Wahrscheinlichkeit mindestens $1/n$ nicht für P_i. Also akzeptiert P_i trotzdem, während ein anderer Prozessor P_j im obigen Sinne *nicht* akzeptiert.

Deshalb schwächt man für die folgenden Empfänger die Akzeptionsbedingung ab und führt λ **Akzeptionsstufen** ein, in [ChRo_91] speziell $\lambda = 2$:[28]

- Findet P_i *alle* seine Minisignaturen korrekt vor, so 1-*akzeptiert* er ψ.
- Findet P_i wenigstens *eine* korrekte Minisignatur, so 2-*akzeptiert* er ψ.
- Ansonsten weist er ψ zurück.

Betrachtet man nur zwei korrekte Prozessoren P_i und P_j, so ist die Wahrscheinlichkeit dafür, daß P_j eine Pseudosignatur *nicht* 2-akzeptieren wird, obwohl P_i sie 1-akzeptiert, höchstens 2^{-m} (weil man dazu m mal richtig raten müßte, welcher Testschlüssel von P_i und welcher von P_j ist). Die Wahrscheinlichkeit, daß es irgendeinen korrekten Prozessor P_j gibt, an den P_i die Pseudosignatur nicht weitergeben kann, ist also kleiner als $n2^{-m}$.

Pseudosignaturen sind im wesentlich ebenso schwer zu fälschen wie die Minisignaturen: Damit eine Fälschung von P_i wenigstens 2-akzeptiert wird, muß wenigstens eine von mn Minisignaturen erfolgreich gefälscht sein. Können Minisignaturen mit Wahrscheinlichkeit $\leq 2^{-\tau}$ gefälscht werden, so gelingt dies für Pseudosignaturen mit Wahrscheinlichkeit $\leq mn2^{-\tau}$.

Im folgenden werden die noch fehlenden Teile des Pseudosignaturschemas ergänzt, die dargestellten Pseudosignaturen aus [ChRo_91] von 2 Akzeptionsstufen auf λ erweitert (später ist $\lambda = t+1$) und die Sicherheit des Pseudosignaturschemas bewiesen.

In Abschnitt 3.1.2 werden Minisignaturen für beliebig lange Nachrichten im

---

[28] In [ChRo_91] wird für 2-Akzeption gefordert, daß $m/2$ Tests erfüllt werden. Dies führt wohl zu etwas kürzeren Signaturen, macht die Darstellung aber unanschaulicher.

byzantinischen Fehlfunktionsmodell mit Authentikationscodes realisiert. Abschnitt 3.1.3 beschreibt das Initialisierungsprotokoll, Abschnitt 3.1.4 die Erweiterung auf λ Akzeptionsstufen.

Die Sicherheit wird zum Schluß betrachtet. Abschnitt 3.1.5 bestimmt zunächst die Erfolgswahrscheinlichkeit eines *passiven* Angriffs. Für Pseudosignaturen sind aber auch *aktive* Angriffe zu betrachten: Angenommen, die Störer könnten eine Pseudosignatur ψ mehrfach an P_i schicken und P_i würde jedesmal erkennen lassen, ob er sie 1-akzeptiert (im obigen Sinne). Lassen die Störer aus ψ nacheinander jede Minisignatur einmal weg, so erfahren sie genau, welche Minisignaturen zu P_i gehören, und können damit eine Pseudosignatur bilden, die zwar alle Prozessoren 1-akzeptieren, P_i aber nicht einmal 2-akzeptiert. Die Möglichkeit solcher Angriffe hängt vom Protokoll ab, in dem die Pseudosignaturen eingesetzt werden. In Abschnitt 3.1.6 wird für polynomiale Protokolle, die gewissen plausiblen Bedingungen genügen, gezeigt, daß in ihnen Pseudosignaturen ebenfalls nur mit exponentiell kleiner Erfolgswahrscheinlichkeit angegriffen werden können.

3.1.2 Minisignaturen durch Authentikationscodes

Eine Minisignatur muß genau einen Prozessor überzeugen und im byzantinischen Fehlfunktionsmodell fälschungssicher sein.

Dies ist genau die Spezifikation eines **Authentikationscodes** [GiMS_74]: Sei hierzu $Auth$ der Codierungsalgorithmus und τ ein Sicherheitsparameter. Können Signierer S und Empfänger E a priori nichtabhörbar und sicher einen *Authentikationsschlüssel* T austauschen, so wird E anhand von

$$(N, Auth(T, N))$$

N stets akzeptieren, während die Störer selbst mit Kenntnis von $(N, Auth(T, N))$ den Empfänger E nur mit Wahrscheinlichkeit $\leq 2^{-\tau}$ zur Akzeption einer Nachricht $N' \neq N$ bewegen können, d. h. $Auth(T, N')$ erraten können.

Man sieht sofort, daß es hierzu mindestens 2^{τ} mögliche Schlüssel geben muß. Zudem darf ein Authentikationsschlüssel nur *einmal* verwendet werden, d. h. um u Nachrichten zu signieren, müssen S und E a priori u Schlüssel austauschen.

In [ChRo_91] wird ein sehr einfacher Authentikationscode mit Paßwörtern für 1-bit-Nachrichten verwendet. Zur Authentikation einer z-bit-Nachricht N mit Fehlerwahrscheinlichkeit $\leq 2^{-\tau}$ benötigt das Verfahren wenigstens $2z\tau$ Schlüsselbits. Eine authentisierte Nachricht, also $(N, Auth(T,N))$, besteht aus wenigstens $z\tau$ bit.

Ein effizienterer Authentikationscode für z-bit-Nachrichten wird in [WeCa_81: §3] beschrieben: Für eine Fehlerwahrscheinlichkeit von $2^{-\tau}$ werden weniger als

$$\alpha(z,\tau) := 4\lceil \log_2(\frac{z}{\tau+1})\rceil(\tau + 1 + \lceil \log_2(\log_2(z))\rceil)$$

Schlüsselbits benötigt.[29] Eine authentisierte Nachricht besteht aus $z+\tau+1$ Bits (z für die eigentliche Nachricht N, $\tau+1$ für den Authentikator $Auth(T, N)$). Die Berechnung von $Auth(T, N)$ ist polynomial in z und τ.

Im folgenden werden die Testschlüssel $T_{j,i}$ stets solche Authentikationsschlüssel der Länge $\alpha(z,\tau)$ sein.

3.1.3 Das Initialisierungsprotokoll für Pseudosignaturen

Das Initialisierungsprotokoll für eine Pseudosignatur besteht aus m Iterationen eines Grundinitialisierungsprotokolls.

In diesem wählt jeder Prozessor P_i einen Authentikationsschlüssel T_i, und alle gemeinsam senden die Menge $\{T_1, \ldots, T_n\}$ an den Signierer S. Die Übertragung von T_i an S muß in perfekter Geheimhaltung und Authentizität erfolgen; sonst ist nicht sicher, daß S als einziger signieren kann bzw. daß P_i die korrekte Pseudosignatur akzeptiert. Die Zuordnung der Authentikationsschlüssel zu den einzelnen Prozessoren muß vor den Störern verborgen bleiben; sonst ist die Weitergebbarkeit nicht gesichert.

Das folgende Grundinitialisierungsprotokoll unterscheidet sich (aus Effizienz- und Beweisgründen) von dem in [ChRo_91]. Die Grundidee ist jedoch dieselbe: man verwendet ein Protokoll zum anonymen Senden, das *DC-Protokoll* [Chau_88], in Kombination mit einem Protokoll zum Mehrfachzugriff [BoBo_90].

3.1.3.1 Das DC-Protokoll

Sei F = GF(2^u) der endliche Körper mit 2^u Elementen für ein hinreichend großes u. Jeder Prozessor P_i besitze als lokale Eingabe einen Wert $x_i \in F$. (Im folgenden wird x_i im wesentlichen ein von P_i zufällig gewählter Authentikationsschlüssel sein. Deshalb wird $u = \alpha(z,\tau)$ gewählt werden.)

Mit CHAUMs **DC-Protokoll**[30] wird die Summe Σ der x_i so berechnet, daß die Störer über die einzelnen x_i außer dieser Summe keine zusätzliche Information erhalten [Chau_85, Chau_88].

---

[29] Dies folgt direkt aus [WeCa_81: §3] mit $A' = \{0,1\}^z$ und $B' = \{0,1\}^{\tau+1}$, wenn man die Konstruktion dort etwas genauer nachrechnet.

[30] Der Name *DC-Protokoll* entstammt einem motivierenden Beispiel aus [Chau_88], den *Dining Cryptographers*.

Protokoll 3.1: CHAUMs DC-Protokoll mit Parameter u

Sei $F = GF(2^u)$.

Phase 1: Schlüsselaustausch

Für alle Paare (P_i, P_j) von Prozessoren mit $i < j$:

P_i wählt zufällig einen **DC-Schlüssel** $K_{i,j} \in F$ und sendet ihn über eine sichere, nichtabhörbare Verbindung an P_j.

P_j nennt den empfangenen DC-Schlüssel $K_{j,i}$. Erhält er keinen DC-Schlüssel von P_i, so setzt er $K_{j,i} := 0$.

Phase 2: DC-Runde

Jeder Prozessor P_i hat als lokale Eingabe für diese Phase eine **DC-Nachricht** $x_i \in F$.

Jeder Prozessor P_i berechnet seine **lokale Summe**

$$O_i := x_i + K_{i,1} + \ldots + K_{i,n}$$

und sendet sie an alle Prozessoren.

Jeder Prozessor berechnet lokal die **globale Summe**

$$\Sigma := O_1 + \ldots + O_n.$$

Phase 1 ist unabhängig von den Werten x_i und kann daher auch in ein Vorbereitungsprotokoll verlegt werden. Phase 2, die DC-Runde, benötigt nur einen Verteilkanal. Das folgende Lemma 3.1 gilt auch dann, wenn dieser Verteilkanal unzuverlässig ist. Lemma 3.1c wird sichern, daß nur der beabsichtigte Signierer S die Authentikationsschlüssel erhält.

Lemma 3.1: (CHAUMs DC-Protokoll)

a) Werden in Phase 2 alle lokalen Summen korrekt gebildet, so ist
$$\Sigma = x_1 + \ldots + x_n.$$

b) Die Störer erhalten, unabhängig von ihrem Verhalten in Phase 1 und Phase 2, über die lokalen Eingaben der korrekten Prozessoren keine Information außer ihrer Summe.[31]

---

31 Formaler: Seien o. B. d. A. $P_1, \ldots, P_k$ die korrekten Prozessoren und $W(x_1, \ldots, x_k)$ die den Störern a priori bekannte Verteilung ihrer lokalen Eingaben $(x_1, \ldots, x_k)$. In einem Protokollablauf besteht die *Beobachtung* der Störer aus den ihnen bekannten DC-Schlüsseln und den lokalen Ausgaben der korrekten Prozessoren. Man sieht leicht, daß sie hieraus die Summe Σ^* der lokalen Eingaben der korrekten Prozessoren berechnen können. Lemma 3.1b besagt dann, daß für jedes Tupel $(x_1, \ldots, x_k)$ gilt: $W(x_1, \ldots, x_k \mid \textit{Beobachtung}) = W(x_1, \ldots, x_k \mid \Sigma^*)$.

c) Veröffentlicht in Phase 2 Prozessor P_i als einziger *nicht* seine lokale Summe, oder statt der korrekten lokalen Summe eine zufällig aus F gewählte, so kann er dennoch die globale Summe Σ berechnen. Alle anderen erhalten dann gar keine Information über die lokalen Eingaben.

d) Der Rechenaufwand des Protokolls ist polynomial in n und u.

In Phase 1 werden in einer Runde in $< n^2$ Nachrichten $< n^2u$ bit über sichere, nichtabhörbare Verbindungen gesendet.

In Phase 2 werden in einer Runde in n Nachrichten nu bit verteilt. ♦

Beweis:

a) In Σ wird jeder DC-Schlüssel $K_{i,j} = K_{j,i}$ genau zweimal addiert, und in GF(2^u) ist dann $K_{i,j} + K_{j,i} = 0$. Folglich ist $\Sigma = x_1 + \ldots + x_n$.

b) [Chau_88].

c) P_i kann, da er alle O_j kennt, Σ berechnen. Einem zufällig aus F gewählten O_i entspricht ein in F gleichverteiltes x_i. Sendet daher P_i ein solches O_i, so erhalten nach (b) die Störer lediglich die Summe $x_1 + \ldots + x_n$, und die ist in F gleichverteilt, enthält also keine Information. Dasselbe gilt erst recht dann, wenn P_i keine lokale Summe O_i versendet.

d) Einfaches Nachrechnen. ❒

Lemma 3.1 gilt auch unter etwas schwächeren Voraussetzungen:

Statt F = GF(2^u) kann jede endliche kommutative (ABELsche) Gruppe verwendet werden [Pfit_90]. In Phase 1 muß der Empfänger P_j des Schlüssels $K_{i,j}$ dann nur $K_{j,i} := -K_{i,j}$ setzen. In Abschnitt 3.1.3.2 muß F allerdings ein endlicher Körper sein, und in Abschnitt 3.1.3.3 muß er die Charakteristik 2 haben.

Ist anfangs einiges über die Fehlersituation bekannt, so müssen zum Beweis von Lemma 3.1 nicht unbedingt alle Prozessoren in Phase 1 einen Schlüssel austauschen. Es genügt, wenn alle korrekten Prozessoren P_i, P_j bezüglich ihrer DC-Schlüssel *zusammenhängen*, d. h. wenn es eine Folge $P_{i_1}, \ldots, P_{i_k}$ korrekter Prozessoren gibt, so daß gilt $P_{i_1} = P_i$, $P_{i_k} = P_j$ und für $h = 1, \ldots, k-1$ haben P_{i_h} und $P_{i_{h+1}}$ einen DC-Schlüssel ausgetauscht [Chau_88].

3.1.3.2 Mehrfachzugriff für das DC-Protokoll nach BOS / DEN BOER

Das DC-Protokoll realisiert einen additiven Kanal mit Kollisionen:

Sind alle DC-Nachrichten $x_i = 0$ bis auf eine, x_j, so ist im ungestörten Fall $\Sigma = x_j$ und die DC-Nachricht x_j wird *erfolgreich* übertragen. Ansonsten tritt eine *Kollision* auf.

Da letztendlich alle Authentikationsschlüssel T_i übertragen werden sollen, benötigt man ein passendes Mehrfachzugriffsprotokoll. Im folgenden wird das von BOS und DEN BOER in [BoBo_90] beschriebene verwendet.

Protokoll 3.2: Mehrfachzugriff nach BOS / DEN BOER mit Parametern u und γ

Sei $F = \mathrm{GF}(2^u)$. γ bestimmt die Fehlerwahrscheinlichkeit ($\leq 2^{-\gamma}$).

Phase 1: DC-Schlüsselaustausch

Mit Phase 1 von Protokoll 3.1 werden parallel DC-Schlüssel für n DC-Runden mit Parameter u ausgetauscht.

Phase 2: Senden von Nachrichten

[1] Jeder Prozessor P_i hat als lokale Eingabe für diese Runde eine DC-Nachricht $x_i \neq 0$.

Phase 2 von Protokoll 3.1 wird parallel n mal ausgeführt: Für die h-te Ausführung, $h = 1, \ldots, n$, wählt Prozessor P_i als lokale Eingabe

$$x_i^{2h-1}$$

und erhält als globale Summe Σ_{2h-1}.

[2] Aus den globalen Summen $\Sigma_1, \Sigma_3, \ldots, \Sigma_{2n-1}$ berechnet jeder Prozessor, falls möglich, eine Menge $Y = \{y_1, \ldots, y_l\} \subseteq F \setminus \{0\}$ mit $l \leq n$, so daß für $h = 1, \ldots, n$ gilt

$$\Sigma_{2h-1} = y_1^{2h-1} + \ldots + y_l^{2h-1}.$$

(Das Verfahren hierfür ist in [BoBo_90, Burt_71, Rabi_80] beschrieben und im Beweis von Lemma 3.2 skizziert. Seine Fehlerwahrscheinlichkeit ist $\leq 2^{-\gamma}$.)

Phase 2, Schritt [2] basiert auf folgendem Lemma:

Lemma 3.2: (Potenzsummen)

Sei $F = \mathrm{GF}(2^u)$ für ein $u \in \mathbb{N}$, und sei $k \in \mathbb{N}$. Eine Menge $X = \{x_1, \ldots, x_l\} \subseteq F \setminus \{0\}$ mit $l \leq k$ heiße **Lösung** von $\Sigma = (\Sigma_1, \Sigma_3, \ldots, \Sigma_{2k-1}) \in F^k$, wenn für $h = 1, \ldots, k$ gilt

$$\Sigma_{2h-1} = x_1^{2h-1} + \ldots + x_l^{2h-1}.$$

a) Jedes $\Sigma \in F^k$ hat höchstens eine Lösung. Es gibt Σ, die keine Lösung haben.

b) Ist für ein Tupel $(x_1, \ldots, x_k) \in F^k$ das Tupel Σ *definiert* durch

$$\Sigma_{2h-1} := x_1^{2h-1} + \ldots + x_k^{2h-1},$$

so ist die einzige Lösung von Σ die Menge X aller $x_i \neq 0$, die in $(x_1, \ldots, x_k)$ in ungerader Anzahl vorkommen.

c) Sei γ ein Sicherheitsparameter. Dann gibt es einen in γ, k und u polynomialen Algorithmus A, der zu jeder Eingabe Σ als Ausgabe entweder eine Lösung oder die Ausgabe „$\varnothing$“ liefert.

Für jedes Σ gilt: Existiert eine Lösung von Σ, so gibt der Algorithmus mit Wahrscheinlichkeit $\geq 1 - 2^{-\gamma}$ diese Lösung aus. ♦

Lemma 3.2b beschreibt den korrekten Fall, in dem sich in Phase 2 alle Prozessoren an das Protokoll halten. Haben alle Prozessoren verschiedene lokale Eingaben $x_i \neq 0$ gewählt, so wird in Phase 2, Schritt [2] gerade die Menge all dieser lokalen Eingaben berechnet.

Halten sich die Störer nicht an das Protokoll, so kann in Phase 2, Schritt [1] ein beliebiges Tupel Σ entstehen. Nach Lemma 3.2a kann es dann passieren, daß in Schritt [2] *keine* Lösung für Σ gefunden wird.

Beweis:

a) Sei $X = \{x_1, \ldots, x_l\}$ eine Lösung von Σ. Zu zeigen ist, daß dann X die einzige Lösung von Σ ist. Man betrachte hierzu das Polynom

$$\sigma(Z) := (1-Zx_1) \ldots (1-Zx_l) =: \sum_{i=0}^{l} \sigma_i Z^i,$$

dessen Nullstellen gerade die Werte x_i^{-1} sind.

Dieses Polynom ist durch Σ eindeutig bestimmt, und damit ist auch die Nullstellenmenge von σ, d. h. X, eindeutig durch Σ bestimmt: Für die Koeffizienten σ_i gelten die NEWTONschen Gleichungen [Burt_71]

$$\Sigma_h + \sigma_1\Sigma_{h-1} + \ldots + \sigma_{h-1}\Sigma_1 + \sigma_h \bullet h = 0, \; h = 1, 2, \ldots,$$

wobei $\Sigma_h = {x_1}^h + \ldots + {x_l}^h$ auch für gerade h und $\sigma_h \bullet h := \sigma_h + \ldots + \sigma_h$ (h Summanden). Nach [Burt_71] bestimmen in GF(2^u) die Gleichungen für $h = 1, 2, \ldots, 2k-1$ eindeutig die $\sigma_1, \ldots, \sigma_k$. Σ genügt zur Bestimmung der nötigen NEWTONschen Gleichungen, denn es ist $\Sigma_{2h} = {\Sigma_h}^2$ (wegen $(a+b)^2 = a^2+b^2$ in GF(2^u)).

Folglich bestimmt jedes Σ genau ein Polynom $\sigma(Z)$. Zerfällt $\sigma(Z)$ vollständig in Linearfaktoren, so ist diese Zerlegung bis auf die Reihenfolge der Faktoren eindeutig. Es gibt daher höchstens eine Lösung X, so daß $\sigma(Z) = (1-Zx_1) \ldots (1-Zx_l)$.

Es gibt Tupel Σ, die keine Lösung haben:

Sei hierzu L die Menge der $X \subseteq F\setminus\{0\}$ mit $|X| \leq k$, d. h. der syntaktisch möglichen Lösungen. Offensichtlich bestimmt jedes $X \in L$ eindeutig das Tupel Σ, dessen Lösung es ist. Gäbe es für jedes Σ eine Lösung, so müßte daher gelten $|F^k| = |L|$. Man sieht aber leicht, daß $|F^k| > |L|$: Zu $\Sigma = (\Sigma_1, \Sigma_3, \ldots, \Sigma_{2k-1})$ sei $S(\Sigma) := \{\Sigma_1, \Sigma_3, \ldots, \Sigma_{2k-1}\}\setminus\{0\}$. Offensichtlich ist $S(\Sigma) \in L$. Die Abbildung $S: F^k \to L$ ist surjektiv, denn für jedes $X = \{x_1, \ldots, x_l\} \in L$ ist $S(x_1, \ldots, x_l, 0, \ldots, 0) = X$, und sie ist nicht injektiv, denn $S(\Sigma)$ ist unabhängig von der Reihenfolge der Elemente in Σ. Also ist $|F^k| > L$.

b) Man sieht leicht, daß X eine, und wegen (a) die einzige Lösung von Σ ist: In F gilt $x+x = 0$. Tritt daher in $(x_1, \ldots, x_k)$ ein x_i in gerader Anzahl auf, so addieren sich alle Summanden ${x_i}^h$ in Σ_h zu 0, d. h. man kann sie weglassen. Tritt x_i in ungerader Anzahl g auf, so addieren sich die ersten $g-1$ Summanden ${x_i}^h$ in Σ_h wieder zu 0, d. h. man kann alle g Summanden durch einen ersetzen. Ein $x_i = 0$

macht sich in Σ_h überhaupt nicht bemerkbar.

c) Zu gegebenem Σ werden zunächst wie in (a) die ersten $2k$–1 NEWTONschen Gleichungen aufgestellt, und nach [Burt_71] kann hieraus in in k und u polynomialer Zeit $\sigma(Z)$ bestimmt werden.

Existiert eine Lösung X, so ist sie gerade die Menge der Inversen der Nullstellen von $\sigma(Z)$, d. h. man muß $\sigma(Z)$ faktorisieren, wozu ein Algorithmus von MICHAEL RABIN verwendet wird [Rabi_80: §2 und §5, Th. 6]:

Der Algorithmus erhält als Eingabe ein Polynom $f(Z)$, das in F vollständig in Linearfaktoren zerfällt. Ist $f(Z) = aZ+b$, so wird die Nullstelle $Z = -b/a$ ausgegeben. Ansonsten zerlegt der Algorithmus $f(Z)$ in zwei nichttriviale Faktoren $f_0(Z)$ und $f_1(Z)$ und wendet sich rekursiv auf $f_0(Z)$ und $f_1(Z)$ an.

Die Zerlegung von $f(Z)$ in $f_0(Z)$ und $f_1(Z)$ erfolgt durch einen probabilistischen Algorithmus, der für jedes Polynom $f(Z)$ mit Wahrscheinlichkeit $\geq 1/2$ erfolgreich ist [Rabi_80: Th. 6]. Bricht man nach $\lceil \gamma+\log_2(k) \rceil$ erfolglosen Zerlegungsversuchen den Algorithmus ab, so erreicht man eine Fehlerwahrscheinlichkeit $\leq 2^{-\gamma-\log_2(k)} = k^{-1}2^{-\gamma}$. Hat $f(Z)$ den Grad k, so wird für eine Faktorisierung die Zerlegung $< k$ mal aufgerufen, so daß die Fehlerwahrscheinlichkeit der gesamten Faktorisierung $\leq 2^{-\gamma}$ ist.

Der Aufwand von RABINs Algorithmus ist dann polynomial in u, k und γ.[32]

Bricht bei Eingabe $\sigma(Z)$ RABINs Algorithmus erfolglos ab, so gibt der in (c) gesuchte Algorithmus „$\emptyset$“ aus. Ansonsten wird geprüft, ob die Menge der Inversen der von RABINs Algorithmus gefundenen Nullstellen $\neq 0$ eine Lösung sind. Wenn ja, so wird diese Menge ausgegeben, sonst „$\emptyset$“.

Der Gesamtfehler ist gerade der von RABINs Algorithmus, also $\leq 2^{-\gamma}$. ❒

Lemma 3.3: (Mehrfachzugriff für das DC-Protokoll nach BOS / DEN BOER)

a) Führen alle Prozessoren Phase 2 korrekt aus, so erhalten mit Wahrscheinlichkeit $\geq 1-n2^{-\gamma}$ in Schritt [2] alle Prozessoren eine Menge Y. Diese ist dann gerade die Menge aller lokalen Eingaben $x_i \neq 0$, die in $(x_1, \ldots, x_n)$ mit ungerader Vielfachheit enthalten sind.

b) Sei X' die Vielfachmenge der lokalen Eingaben der korrekten Prozessoren und Y' die Menge alle $x_i \in X'$, die in X' mit ungerader Vielfachheit enthalten sind. Dann erhalten die Störer, unabhängig von ihrem Verhalten in Phase 1 und Phase 2, über die lokalen Eingaben der korrekten Prozessoren keine Information außer Y'.

c) Veröffentlicht in Phase 2 Prozessor P_i als einziger *nicht* seine lokalen

---

32 In [Rabi_80] ist der Aufwand nur erwartet polynomial, da keine polynomiale Schranke für die Anzahl der Zerlegungsversuche verwendet wird. Für die hiesige synchrone Rundeneinteilung ist es aber notwendig, eine deterministische polynomiale Zeitschranke für jede Runde zu kennen.

Summen, oder statt der korrekten lokalen Summen zufällig aus F gewählte, so kann er dennoch alle globalen Summen berechnen und Schritt [2] ausführen. Alle anderen erhalten dann gar keine Information über die lokalen Eingaben.

d) Der Rechenaufwand des Protokolls ist polynomial in n, u und γ.

In Phase 1 werden in einer Runde in $< n^2$ Nachrichten $< n^3u$ bit über sichere, nichtabhörbare Verbindungen gesendet.

In Phase 2 werden in einer Runde in n Nachrichten n^2u bit verteilt. ♦

Beweis:

a) Existiert eine Lösung, so erhält sie nach Lemma 3.2c jeder korrekte Prozessor mit Wahrscheinlichkeit $\geq 1–2^{-\gamma}$, alle zusammen also mit Wahrscheinlichkeit $\geq 1–n2^{-\gamma}$. Der Rest folgt aus Lemma 3.2b.

b) Seien o. B. d. A. $P_1, \ldots, P_k$ die korrekten Prozessoren. Nach Lemma 3.1b erhalten die Störer außer den Potenzsummen $\Sigma^*_{2h-1} = {x_1}^{2h-1} + \ldots + {x_k}^{2h-1}$, $h = 1, \ldots, n$, keine Information über $(x_1, \ldots, x_k)$. Da jede Permutation von $(x_1, \ldots, x_k)$ zu denselben Potenzsummen führt, erfahren die Störer höchstens jedes x_i, erhalten aber keine Information darüber, welches x_i von welchem Prozessor stammt. Da in GF(2^u) für alle x gilt $x + x = 0$, addieren sich in den Potenzsummen alle lokalen Eingaben, die von einer geraden Anzahl korrekter Prozessoren gewählt wurden, zu 0. Folglich erhalten die Störer diese lokalen Eingaben nicht. Übrig bleibt die Menge Y'.

c) Lemma 3.1c.

d) Lemma 3.1d (jeweils für n parallele Ausführungen) und Lemma 3.2c. ❒

Lemma 3.3a ist für den korrekten Fall das beste zu erwartende Ergebnis, denn DC-Nachrichten x, die eine gerade Anzahl von Prozessoren als lokale Eingaben wählen, löschen sich in den Summen Σ_{2h-1} wegen $x + x = 0$ gegenseitig aus.

Im folgenden wird dies kein Problem sein: Alle korrekten Prozessoren werden ihre lokalen Eingaben x_i zufällig aus F wählen, und F wird so groß sein, daß fast sicher alle korrekten Prozessoren verschiedene lokale Eingaben verwenden.

3.1.3.3 Initialisierungsprotokoll

Geht man davon aus, daß alle Prozessoren sich an der Grundinitialisierung korrekt beteiligen, so ist das Grundinitialisierungsprotokoll recht einfach:

Alle Prozessoren P_i senden mit Protokoll 3.2 ihren Authentikationsschlüssel T_i an S. In den DC-Runden gibt S keine (oder zufällig gewählte) lokale Summen aus und erhält somit als einziger die Authentikationsschlüssel.

Geht man von Störern aus, so genügt dies aber noch nicht ganz: Man muß noch sicherstellen, daß, wenn S und P_i korrekt sind, der Authentikations-

schlüssel von P_i trotz eventueller Störungen bei S ankommt, damit P_i später eine Pseudosignatur von S sicher akzeptiert.

Zunächst weiß man aus Lemma 3.3a, daß, wenn zwei Prozessoren *identische* Authentikationsschlüssel wählen, S diese nicht erhält.

Ist S korrekt, so ist es aber fast unmöglich, daß ein korrekter Prozessor P_i denselben Authentikationsschlüssel verwendet wie ein anderer (fehlerhafter oder korrekter) Prozessor: Mit Lemma 3.3c wird erreicht, daß S während der Initialisierung als einziger Information über die Authentikationsschlüssel erhält. Alle Prozessoren müssen daher ihre Authentikationsschlüssel unabhängig von dem von P_i wählen, und es gibt wenigstens 2^τ mögliche Authentikationsschlüssel. Die Wahrscheinlichkeit, daß es zwei korrekte Prozessoren gibt, die denselben Authentikationsschlüssel wählen, ist daher $\leq n^2 2^{-\tau}$.

Sehr viel ernsthafter stört ein Prozessor, der statt der ersten n ungeraden Potenzen seiner lokalen Eingabe beliebige andere Werte verwendet. Hier sind zwei Probleme zu lösen:

1. Wie kann S eine solche Störung sicher erkennen?
2. Wie werden Störungen beseitigt?

Zur Erinnerung: Es wird angenommen, daß im Grundinitialisierungsprotokoll ein zuverlässiger Verteilkanal verwendet wird.

3.1.3.3.1 Erkennen von Störungen im DC-Protokoll

Um Störungen zu erkennen, werden nicht nur (wie in Protokoll 3.2) die ersten n, sondern die ersten $2n$ ungeraden Potenzsummen verwendet:

Protokoll 3.3: Grundinitialisierungsprotokoll mit Störungserkennung mit Parametern z, τ und γ.

z ist die Länge der zukünftig zu signierenden Nachricht, τ der Sicherheitsparameter des verwendeten Authentikationscodes, γ der von Protokoll 3.2.

Sei $u := \alpha(z,\tau)+1$. u ist so groß gewählt, daß mit $F = \mathrm{GF}(2^u)$ die Authentikationsschlüssel eindeutig durch Elemente aus $F\backslash\{0\}$ dargestellt werden können.

Sei S der künftige Signierer, d. h. Empfänger der Authentikationsschlüssel.

Phase 1: DC-Schlüsselaustausch

Mit Phase 1 von Protokoll 3.1 werden parallel DC-Schlüssel für $2n$ DC-Runden mit Parameter u ausgetauscht.

Phase 2: Senden der Authentikationsschlüssel

[1] Jeder Prozessor P_i wählt zufällig einen Authentikationsschlüssel T_i, codiert als ein Element aus $F\backslash\{0\}$.

Phase 2 von Protokoll 3.1 wird parallel $2n$ mal ausgeführt, wobei jedoch der Signierer S seine lokalen Summen nicht ausgibt. Für die h-te Ausführung, $h = 1, \ldots, 2n$, wählt Prozessor P_i als lokale Eingabe

$$T_i^{2h-1}.$$

S erhält aus den $2n$ DC-Runden die globalen Summen $\Sigma_1, \Sigma_3, \ldots, \Sigma_{4n-1}$.

[2] Aus den globalen Summen $\Sigma_1, \Sigma_3, \ldots, \Sigma_{2n-1}$ berechnet S, falls möglich, eine Menge $Y = \{y_1, \ldots, y_l\} \subseteq F\backslash\{0\}$ mit $l \leq n$, so daß für $h = 1, \ldots, n$ gilt

$$\Sigma_{2h-1} = y_1^{2h-1} + \ldots + y_l^{2h-1}.$$

(Wie in Protokoll 3.2, Phase 2, Schritt [2].)

Gelingt dies und gilt auch für alle $h = n+1, \ldots, 2n$

$$\Sigma_{2h-1} = y_1^{2h-1} + \ldots + y_l^{2h-1},$$

so **akzeptiert** S, ansonsten veröffentlicht er eine **Beschwerdenachricht**.

Akzeptiert S, so verwendet er als Signierschlüssel (für den hier initialisierten Abschnitt)

$$\{y_1, \ldots, y_l\}.$$

In [ChRo_91] wird zur Erkennung von Störungen ein anderes Verfahren verwendet. Bei Verwendung des effizienteren Authentikationscodes (Abschnitt 3.1.2) ist das dortige Verfahren allerdings nicht effizienter als das hier verwendete, und ein Analogon zu folgendem Lemma 3.4 wurde nicht bewiesen.

Lemma 3.4: (Störungserkennung im Grundinitialisierungsprotokoll)

a) Ist S korrekt, so erhalten die Störer keine Information über die Authentikationsschlüssel T_i der korrekten Prozessoren.

b) Ist S korrekt, so gilt mit Wahrscheinlichkeit $\geq 1 - 5n^2 2^{-\tau-1}$: Sendet S am Ende von Phase 2 keine Beschwerdenachricht, so hat er von jedem korrekten Prozessor dessen Authentikationsschlüssel erhalten.

c) Ist S fehlerhaft und sind o. B. d. A. $P_1, \ldots, P_k$ die korrekten Prozessoren, so erhalten die Störer keine Information über die Zuordnung der Authentikationsschlüssel $\{T_1, \ldots, T_k\}$ zu den korrekten Prozessoren, die sie wählten.

d) Ist S korrekt und gibt es eine Menge $\{y_1, \ldots, y_l\} \subseteq F\backslash\{0\}$ mit $l \leq n$, die

$$\Sigma_{2h-1} = y_1^{2h-1} + \ldots + y_l^{2h-1}, \; h = 1, \ldots, 2n$$

erfüllt, so beschwert sich S dennoch mit Wahrscheinlichkeit $\leq 2^{-\gamma}$.

e) Der Rechenaufwand des Protokolls ist polynomial in n, z, τ und γ.

In Phase 1 werden in einer Runde $O(n^3 \log(z) (\tau + \log(\log(z))))$ bit in

$< n^2$ Nachrichten über sichere, nichtabhörbare Verbindungen gesendet.

In Phase 2 werden in zwei Runden $O\big(n^2 \log(z)\ (\tau + \log(\log(z)))\big)$ bit in n Nachrichten verteilt. ♦

Beweis:

a) Folgt aus Lemma 3.3c.

b) Seien o. B. d. A. die ersten k Prozessoren korrekt, und sei S unter ihnen.

Vorbemerkung: Es wird nur der Fall betrachtet, daß die $T_1, \ldots, T_k$ jeweils paarweise verschieden sind.

Dieser Fall tritt mit Wahrscheinlichkeit

$$\geq 1 - n^2 2^{-\tau-1}$$

ein: Seien P_i, P_j zwei beliebige korrekte Prozessoren. Dann ist, da T_i, T_j zufällig gewählt werden, die Wahrscheinlichkeit für $T_i = T_j$ höchstens $2^{-\tau}$. Da es $< n^2/2$ Paare korrekter Prozessoren gibt, tritt obiger Fall mit Wahrscheinlichkeit $\geq 1 - n^2 2^{-\tau-1}$ ein. ◊

Störer können nur stören, indem sie ihre lokalen Summen falsch wählen. Insgesamt heißt dies, daß sie gemeinsam Werte $\Delta_1, \Delta_3, \ldots$ wählen, so daß S anstelle der korrekten Potenzsumme Σ_{2h-1} die globale Summe

$$T_1^{2h-1} + \ldots + T_k^{2h-1} + \Delta_{2h-1},\ h = 1, \ldots, 2n,$$

erhält. S akzeptiert dies genau dann, wenn es $l \leq n$ paarweise verschiedene Werte $y_1, \ldots, y_l$ gibt, so daß

$$T_1^{2h-1} + \ldots + T_k^{2h-1} + \Delta_{2h-1} = y_1^{2h-1} + \ldots + y_l^{2h-1},\ h = 1, \ldots, 2n,$$

was in $GF(2^u)$ äquivalent ist zu

$$\Delta_{2h-1} = T_1^{2h-1} + \ldots + T_k^{2h-1} + y_1^{2h-1} + \ldots + y_l^{2h-1},\ h = 1, \ldots, 2n. \qquad (3.1)$$

Eine Störung liegt dann vor, wenn wenigstens ein T_{i^*} mit $i^* \leq k$ nicht übertragen wurde, d. h. nicht unter $y_1, \ldots, y_l$ vorkommt. Dies impliziert, daß T_{i^*} in (3.1) genau einmal, also in ungerader Anzahl, enthalten ist.

Nach Lemma 3.2 gibt es höchstens eine Lösung für $(\Delta_1, \Delta_3, \ldots, \Delta_{4n-1})$, d. h. höchstens eine Menge $\{z_1, \ldots, z_{l^*}\} \subseteq F \setminus \{0\}$, $l^* \leq 2n$, so daß gilt

$$\Delta_{2h-1} = z_1^{2h-1} + \ldots + z_{l^*}^{2h-1},\ h = 1, \ldots, 2n.$$

Da die $T_1, \ldots, T_k, y_1, \ldots, y_l$ nach (3.1) bis auf doppelt vorkommende eine Lösung sind, müssen sie die z_i sein. Also befindet sich T_{i^*} im Falle einer *unbemerkten* Störung unter $z_1, \ldots, z_{l^*}$. Nach (a) haben die Störer keine Information über $T_1, \ldots, T_k$, d. h. die Werte Δ_{2h-1} und somit auch $z_1, \ldots, z_{l^*}$ sind unabhängig von $T_1, \ldots, T_k$. (Im Prinzip wurde also gezeigt, daß die Störer, um unentdeckt zu stören, ein T_{i^*} der korrekten Prozessoren raten mußten.)

Die Wahrscheinlichkeit von $z_j = T_{i^*}$ für feste j, i^* ist höchstens $2^{-\tau}$. Da es $\leq 2n^2$ solche Paare gibt, ist die Wahrscheinlichkeit einer unbemerkten Störung in diesem Fall $\leq n^2 2^{-\tau+1}$.

Zusammen mit dem Fehler aus der Vorbemerkung erhält man eine Fehlerwahrscheinlichkeit $\leq 5n^2 2^{-\tau-1}$.

c) Lemma 3.3b.

d) Lemma 3.2c.

e) Folgt aus Lemma 3.3d (wobei zu beachten ist, daß S in Phase 2 keine lokalen Summen veröffentlicht) und der Definition von $\alpha(z,\tau)$ in Abschnitt 3.1.2. ❒

Lemma 3.4 basiert also darauf, daß kein Prozessor außer S Information über die gesendeten Authentikationsschlüssel erhält, und daß für die Störer bezüglich des Authentikationsschlüssels eines korrekten Prozessors wenigstens τ bit Unsicherheit herrscht.

3.1.3.3.2 Fehlerlokalisierung und Rekonfigurierung

Um den Verursacher einer Störung während der Initialisierungsphase zu lokalisieren, wird im wesentlichen dasselbe Protokoll wie in [Chau_88] verwendet. (Man beachte, daß in der Initialisierungsphase ein zuverlässiger Verteilkanal vorausgesetzt ist.)

Protokoll 3.4 kombiniert dieses Protokoll (in Phase 3) mit dem Grundinitialisierungsprotokoll 3.3. Insgesamt erhält man das komplette Initialisierungsprotokoll einer Pseudosignatur mit m Minisignaturen.

In Protokoll 3.4 werden Prozessoren und DC-Schlüssel öffentlich **disqualifiziert**. Ist der DC-Schlüssel zwischen P_i und P_j disqualifiziert, so verwenden danach P_i und P_j nur noch konstant den DC-Schlüssel $K_{i,j} = K_{j,i} = 0$.

Zwei Prozessoren P_i, P_j heißen zu einem bestimmten Zeitpunkt bezüglich der DC-Schlüssel **zusammenhängend**, wenn es eine Folge von nicht disqualifizierten Prozessoren $P_{i_1}, \ldots, P_{i_k}$ gibt, so daß gilt $P_{i_1} = P_i$, $P_{i_k} = P_j$ und für $h = 1, \ldots, k-1$ ist der DC-Schlüssel zwischen P_{i_h} und $P_{i_{h+1}}$ nicht disqualifiziert.

Wie in Kapitel 2 wird im folgenden jeder Prozessor, der sich für *alle* offensichtlich fehlerhaft verhält, disqualifiziert (vgl. Abschnitt 2.1.2).

Protokoll 3.4: Initialisierungsprotokoll für eine Pseudosignatur mit Parametern m, z, τ.

m ist die Vielfachheit der Pseudosignatur (d. h. die Anzahl ihrer Abschnitte), z die Länge der zukünftig zu signierenden Nachricht und τ der Sicherheitsparameter des verwendeten Authentikationscodes. (τ bestimmt zugleich den Wert γ für Protokoll 3.3.) Sei S der künftige Signierer, d. h. Empfänger der Authentikationsschlüssel.

Phase 1 und Phase 2:

Parallel für $s = 1, \ldots, m$ wird Protokoll 3.3 mit Parametern z, τ und $\gamma := \tau$ ausgeführt.

Um Phase 3 ausführen zu können, muß jeder Prozessor die veröffentlichten lokalen Summen aller Prozessoren speichern.

Hat S das Ergebnis von Phase 2 akzeptiert, so endet die s-te Iteration erfolgreich. (Als Ergebnis hat der Signierer eine Menge SSi_s von bis zu n Authentikationsschlüsseln, jeder korrekte Prozessor P_i einen Authentikationsschlüssel $T_{i,s}$, und fast sicher ist $T_{i,s} \in SSi_s$.)

Phase 3: Fehlerlokalisierung und Rekonfigurierung

Parallel für alle nicht erfolgreich beendeten Iterationen s von Protokoll 3.3: Für alle Prozessoren P_i ($P_i = S$ eingeschlossen):

Aufdecken:

P_i veröffentlicht alle in Phase 1 der s-ten Iteration ausgetauschten DC-Schlüssel $K_{i,j}$ und seinen Wert T_i.

Fehlerlokalisierung:

[a] P_i **disqualifiziert** alle Prozessoren P_j, die
- nicht alle geforderten Werte aufdecken, oder
- einen bereits disqualifizierten DC-Schlüssel weiterverwendet haben, oder
- deren aufgedeckte Werte nicht zu ihren aus Phase 2 bereits bekannten lokalen Summen passen.

Letzteres heißt, daß für eine der $2n$ DC-Runden die bekannte lokale Summe von P_j nicht gleich ist der Summe aus den von P_j für diese DC-Runde aufgedeckten DC-Schlüsseln und der entsprechenden Potenz des von P_j aufgedeckten Authentikationsschlüssels.

[b] Für jedes Paar von Prozessoren (P_j, P_k), die bislang noch nicht disqualifiziert wurden, prüft P_i, ob sie sich bezüglich ihrer gemeinsamen DC-Schlüssel $K_{i,j} = K_{j,i}$ einig waren. Falls nicht, so wird deren gemeinsamer DC-Schlüssel **disqualifiziert**.

[c] Hat P_i bislang weder einen Prozessor noch einen DC-Schlüssel disqualifiziert, so **disqualifiziert** er S.

[d] P_i **disqualifiziert** alle Prozessoren P_j, mit denen er bezüglich der DC-Schlüssel nicht zusammenhängend ist.

(Alle korrekten Prozessoren sind sich jeweils einig, welche Prozessoren disqualifiziert sind; vgl. Satz 3.5a3.)

Rekonfigurierung:

P_i akzeptiert in den folgenden DC-Runden von disqualifizierten Prozessoren keine Nachrichten. Ist P_j disqualifiziert, so verwendet P_i für alle folgenden DC-Runden den DC-Schlüssel $K_{i,j} = 0$.

Disqualifiziert P_i seinen DC-Schlüssel mit P_j, so verwendet er für alle folgenden DC-Runden den DC-Schlüssel $K_{i,j} = 0$.

Wurde S nicht disqualifiziert und gibt es noch wenigstens zwei nicht disqualifizierte Prozessoren, so werden alle gescheiterten Iterationen von Protokoll 3.3 wiederholt.

Die Eigenschaften von Protokoll 3.4 werden im folgenden Satz 3.5 zusammengefaßt.

Satz 3.5: (Initialisierungsprotokoll für Pseudosignaturen der Vielfachheit m)

a) Ist der Signierer S korrekt, so gilt mit Wahrscheinlichkeit

$$\geq 1 - mn^4 2^{-\tau+1}:$$

a1) S erhält von jedem korrekten Prozessor P_i dessen m Authentikationsschlüssel

$$STe_i := (T_{i,1}, \ldots, T_{i,m})$$

und verwendet daher einen Signierschlüssel

$$SSi = (SSi_1, \ldots, SSi_m),$$

für den gilt $\{T_{i_1,j}, \ldots, T_{i_k,j}\} \subseteq SSi_j$, wo $P_{i_1}, \ldots, P_{i_k}$ genau die korrekten Prozessoren sind.

a2) Die Störer erhalten keine Information über die STe_i der korrekten Prozessoren P_i.

a3) Korrekte Prozessoren und DC-Schlüssel, die zwei korrekten Prozessoren gemeinsam sind, werden nicht disqualifiziert. Alle korrekten Prozessoren disqualifizieren in Phase 3 dieselben Prozessoren und DC-Schlüssel.

b) Ist S fehlerhaft, so gilt:

b1) Wird S nicht disqualifiziert und daher aus Sicht der korrekten Prozessoren eine Pseudosignatur initialisiert, so erhalten die Störer dennoch für kein $j \in \{1, \ldots, m\}$ Information über die Zuordnung der Authentikationsschlüssel $\{T_{i_1,j}, \ldots, T_{i_k,j}\}$ zu den korrekten Prozessoren, die sie wählten.

b2) Wie (a3), aber ohne Fehlerwahrscheinlichkeit.

c) Das Protokoll terminiert nach weniger als $n^2/2$ Iterationen von Phase 3. Protokoll 3.3 wird dabei weniger als $mn^2/2$ mal ausgeführt.

d) Der Rechenaufwand des Protokolls ist polynomial in m, n, z, τ.

Das Protokoll benötigt im ungestörten Fall 3, im schlimmsten Fall $2n^2$

Runden.

In jeder Ausführung von Phase 1 und 2 werden in $< n^2$ Nachrichten $O(mn^3 \log(z) (\tau + \log(\log(z))))$ bit über sichere, nichtabhörbare Verbindungen gesendet und in n Nachrichten $O(mn^2 \log(z) (\tau + \log(\log(z))))$ bit über den zuverlässigen Verteilkanal veröffentlicht.

In jeder Ausführung von Phase 3 werden n Nachrichten über den zuverlässigen Verteilkanal veröffentlicht. Für jede gestörte Ausführung von Protokoll 3.3 werden dabei $O(n^3 \log(z) (\tau + \log(\log(z))))$ bit veröffentlicht.

(Im schlimmsten Fall, wenn Protokoll 3.3 $mn^2/2$ mal ausgeführt wird, erhöhen sich die Nachrichten- und Bitanzahlen also um den Faktor $O(n^2)$.) ♦

Beweis: Da (a) auf (c) aufbaut, wird zuerst (c) bewiesen:

c) In jeder Iteration von Phase 3 wird in [a] bis [c] wenigstens ein Prozessor oder ein DC-Schlüssel disqualifiziert: Wird in [a] oder [b] keine Disqualifizierung vorgenommen, so wird in [c] S selbst disqualifiziert. Wird ein Prozessor P disqualifiziert, so gibt es wenigstens einen nicht disqualifizierten DC-Schlüssel von P. Dies stimmt vor der ersten Iteration von Phase 3, und es stimmt in allen folgenden, da P sonst schon in der letzten Iteration nach [d] disqualifiziert worden wäre.

Also geht in jeder Phase 3 wenigstens ein DC-Schlüssel verloren. Das Protokoll beginnt mit $< n^2/2$ DC-Schlüsseln, d. h. nach $\leq n^2/2$ Iterationen sind alle DC-Schlüssel disqualifiziert. Wegen [d] sind dann auch alle Prozessoren disqualifiziert und das Protokoll terminiert. (Real endet das Protokoll spätestens dann, wenn alle Störer disqualifiziert sind.)

Im schlimmsten Fall betrifft jede Störung alle m Ausführungen von Protokoll 3.3 in den Phasen 1 und 2. Folglich muß im schlimmsten Fall Protokoll 3.3 nach jeder der $< n^2/2$ Iterationen m mal ausgeführt werden.

a) Vorbemerkung: Es wird nur der Fall betrachtet, daß für alle nach (c) $\leq mn^2/2$ Ausführungen von Protokoll 3.3 die Aussagen von Lemma 3.4b und Lemma 3.4d fehlerfrei zutreffen. Mit $\gamma = \tau$ ist dies mit Wahrscheinlichkeit (für $n \geq 2$)

$$\leq mn^2 2^{-\tau-1} + mn^4 2^{-\tau+1} \leq mn^4 2^{-\tau+1}$$

nicht der Fall. Dies ist zugleich die Fehlerwahrscheinlichkeit von (a). ◊

a3) Angenommen, P_i oder der DC-Schlüssel zwischen P_i und P_j werden disqualifiziert. Es werden der Reihe nach die 4 Möglichkeiten behandelt.

[a] P_i ist offensichtlich fehlerhaft.

[b] Wären beide Prozessoren korrekt, so hätten sie für alle DC-Runden dieselben DC-Schlüssel verwendet und veröffentlicht. Also ist einer von beiden fehlerhaft (welcher, kann allerdings nicht entschieden werden).

[c] Angenommen, S sei korrekt und werde dennoch disqualifiziert. Dann

paßten alle veröffentlichten Daten zusammen, d. h. sind o. B. d. A. $P_1, \ldots, P_{n'}$ die noch nicht disqualifizierten Prozessoren, so muß S in Phase 2 die Potenzsummen

$$\Sigma_{2h-1} = T_1{}^{2h-1} + \ldots + T_{n'}{}^{2h-1},\ h = 1, \ldots, 2n$$

für die in Phase 3 veröffentlichten T_i erhalten haben, im Widerspruch zu Lemma 3.4d.

[d] Zu zeigen ist, daß alle korrekten Prozessoren stets bezüglich ihrer DC-Schlüssel zusammenhängen.

Bevor Phase 3 das erste Mal ausgeführt wird, ist dies klar, denn alle Prozessoren hängen direkt durch einen DC-Schlüssel zusammen.

In [a] bis [c] werden (wie gezeigt) nur Störer oder DC-Schlüssel disqualifiziert, die Störer mit anderen Prozessoren verbanden. Folglich bleiben die korrekten Prozessoren (sogar direkt) zusammenhängend, und in [d] werden nur Störer disqualifiziert.

Alle korrekten Prozessoren sind sich stets einig, welche Prozessoren und DC-Schlüssel disqualifiziert sind und welche nicht:

Die Disqualifizierungen nach [a] und [b] hängen ausschließlich von veröffentlichter (also zuverlässig verteilter) Information ab, so daß alle korrekten Prozessoren gleich entscheiden. Hieraus folgt, daß auch in [c] alle korrekten Prozessoren gleich entscheiden. Für [d] wurde schon gezeigt, daß stets alle korrekten Prozessoren bezüglich ihrer DC-Schlüssel zusammenhängen, d. h. hängt ein korrekter Prozessor nicht mehr mit P_i zusammen, so hängt *kein* korrekter Prozessor mit P_i zusammen, und *alle* korrekten Prozessoren werden P_i nach [d] disqualifizieren.

a1) Terminiert das Protokoll, so ist nach (a3) kein korrekter Prozessor disqualifiziert und S hat in der letzten Iteration keine Beschwerdenachricht gesendet. Der Rest folgt aus Lemma 3.4b.

a2) Nach (a3) betreffen die Disqualifizierungen nur die Störer. Insbesondere werden nur solche DC-Schlüssel disqualifiziert, die den Störern ohnehin schon bekannt sind. Also gilt nach wie vor Lemma 3.4a.

b1) Ist S nach Protokoll 3.4 nicht disqualifiziert, so wurde aus Sicht der korrekten Prozessoren eine Pseudosignatur initialisiert (ist S disqualifiziert, so stellt sich das Problem erst gar nicht). Für die m Ausführungen von Protokoll 3.3, die S akzeptierte, d. h. die nicht aufgedeckt wurden und daher die Schlüssel STe der korrekten Prozessoren definieren, gilt Lemma 3.4c.

b2) Im Beweis von (a3) war [c] die einzige fehlerbehaftete Disqualifizierungsregel. Der Signierer S ist hier aber als fehlerhaft vorausgesetzt.

d) Rechenaufwand: Die m parallelen Ausführungen von Protokoll 3.3 in Phase 1 und Phase 2 sind nach Lemma 3.4e jeweils polynomial in n, z, τ, und Phase 3 ist offensichtlich polynomial in n, z, τ und m.

Rundenanzahl: Die Phasen 1 und 2 entsprechen denen von Protokoll 3.3, d.h. benötigen insgesamt 3 Runden. In diesen können jeweils alle m Iterationen bzw. alle Iterationen, die nach einer entdeckten Störung wiederholt werden müssen, parallel ausgeführt werden. Im ungestörten Fall benötigt das Protokoll daher 3 Runden.

Phase 3 benötigt eine Runde (für das Aufdecken) und wird nach (d) weniger als $n^2/2$ mal ausgeführt. Damit ergibt sich die maximale Gesamtrundenzahl $2n^2$.

Nachrichtenaufwand: Phasen 1 und 2 benötigen in jeder Iteration maximal den m-fachen Aufwand der entsprechenden Phasen von Protokoll 3.3. Die Nachrichtenanzahl erhöht sich allerdings nicht, da alle Nachrichten, die in einer Runde von P_i an P_j gesendet werden, zu einer zusammengefaßt werden können.

In Phase 3 muß jeder Prozessor eine Nachricht zuverlässig verteilen, alle zusammen also n. Bezogen auf eine gestörte Grundinitialisierung enthält jede Nachricht einen Authentikationsschlüssel (u bit) und DC-Schlüssel für $2n$ Runden, also $< 2n^2u$ bit. Zusammen für alle Prozessoren sind dies $O(n^3 \log(z)\,(\tau + \log(\log(z)))\,)$ bit. ❐

Im allgemeinen werden in einem authentisierten Protokoll mehrere Pseudosignaturen benötigt, d.h. Protokoll 3.4 wird mehrfach ausgeführt, eventuell mit verschiedenen Signierern.

Sinnvollerweise werden dabei alle in irgendeiner Ausführung vorgenommenen Disqualifizierungen für alle nachfolgenden Ausführungen verwendet.

Protokoll 3.4 kann parallel beliebig oft, etwa w mal, ausgeführt werden, ohne daß sich hierfür die Rundenanzahlen erhöhen. Gemessen in bits erhöht sich der Nachrichtenaufwand um den Faktor w. Faßt man alle Nachrichten, die in einer Runde P_i an P_j sendet, zu *einer* zusammen, so erhöht sich die Größenordnung der Nachrichtenanzahl nicht. (Genauer gesagt erhöht sich die Nachrichtenanzahl nur in Phase 2, und zwar von n auf höchstens $2n$. Die $2n$ werden dann erreicht, wenn in den w Ausführungen jeder Prozessor wenigstens einmal als Signierer auftritt, und wenn am Ende jeder Signierer eine Beschwerdenachricht sendet.)

Wird Protokoll 3.4 (in w parallelen Versionen) u mal sequentiell ausgeführt, so sieht man leicht, daß Satz 3.5d analog für alle Ausführungen zusammen gilt, d.h. in allen u Iterationen kann Phase 3 zusammen höchstens $n^2/2$ mal wiederholt werden. Damit benötigen die u Iterationen insgesamt höchstens $m(u+n^2/2)$ Ausführungen von Protokoll 3.3 (bzw. bei jeweils w parallelen Versionen $mw(u+n^2/2)$) und höchstens $3u+2n^2$ Runden.

Hinsichtlich der Parallelisierung von Unterprotokollen gibt es einige sinnvolle Varianten:

Zum einen ist der DC-Schlüsselaustausch in Phase 1 unabhängig von allen anderen Phasen und auch davon, welcher Prozessor als Signierer S agiert. Man kann daher Phase 1 für beliebig viele Ausführungen von Phase 2 parallel auf

Vorrat ausführen. Für u sequentielle Ausführungen von Protokoll 3.4 kann man damit alle Ausführungen von Phase 1 bis auf die erste einsparen, d.h. im besten Fall spart man u–1, im schlimmsten Fall u–1+$n^2/2$ Runden.

Zum anderen führt die Parallelisierung von Protokoll 3.3 in Phase 2 dazu, daß die Störer immer alle parallelen Versionen stören können, ohne hierdurch mehr Prozessoren oder DC-Schlüssel zu verlieren, als wenn sie nur eine Version gestört hätten. Wird Protokoll 3.3 nur sequentiell ausgeführt, so verringert sich daher die Anzahl der nötigen Wiederholungen von Protokoll 3.3 auf $n^2/2$, d.h. in einer Ausführung von $mn^2/2$ auf $m+n^2/2$, in u sequentiellen von $m(u+n^2/2)$ auf $mu+n^2/2$. Die Rundenanzahl steigt dafür allerdings im besten Fall von O(1) auf O(m), im schlimmsten Fall von O(n^2) auf O($m+n^2$).

Generell verringert sich der Aufwand fast aller Protokolle nach jeder Disqualifizierung eines Prozessors, da man dann n durch (n–1) ersetzen kann (und, wo dies eine Rolle spielt, auch t durch t–1).

3.1.4 Pseudosignaturen mit λ Akzeptionsstufen

In Abschnitt 3.1.1 wurden einfache Pseudosignaturen mit 2 Akzeptionsstufen betrachtet [ChRo_91]. Ein Prozessor 1-akzeptierte eine Pseudosignatur, wenn er alle seine Minisignaturen fand, und 2-akzeptierte sie, wenn er wenigstens eine seiner Minisignaturen fand.

Zur Verallgemeinerung von 2 auf λ Akzeptionsstufen wählt man für ein geeignetes (vgl. Schema 3.5 und Abschnitt 3.1.5) Δ die Vielfachheit $m = (\lambda-1)\Delta+1$. Damit ein Prozessor eine Pseudosignatur 1-akzeptiert, wird wieder gefordert, daß er in ihr m Minisignaturen findet. Die schwächeren Akzeptionsstufen sind so definiert, daß, wenn für die k-Akzeption a Minisignaturen genügen, dann $a-\Delta$ für die (k+1)-Akzeption genügen.

Beispiel: Für $\Delta = 2$ und $\lambda = 4$ erhält man $m = 7$, und die Akzeptionsstufen 1 bis 4 entsprechen den Werten $a \in \{7\}, \{7, 6, 5\}, \{7, 6, 5, 4, 3\}, \{7, 6, 5, 4, 3, 2, 1\}$. ✎

Im folgenden Schema 3.5 sind die Pseudosignaturen mit λ Akzeptionsstufen vollständig definiert. Schema 3.5 beschreibt *kein* Protokoll, sondern nur die Unterprotokolle (Phasen), die dann in authentisierten Protokollen benötigt werden.

Schema 3.5: Pseudosignaturen mit Parametern λ, z, τ und κ

λ ist die Anzahl der Akzeptionsstufen einer Pseudosignatur und z die Länge der zukünftig zu signierenden Nachricht. τ ist der Sicherheitsparameter des verwendeten Authentikationscodes und bestimmt die *Korrektheit* und *Unfälschbarkeit*, κ ist der Sicherheitsparameter für die *Weitergebbarkeit*.

Zu den Parametern λ, κ seien die **Akzeptionsdifferenz** Δ und die **Vielfachheit** m definiert durch

$$\Delta := \lceil 2 \ln(2)\, \lambda\kappa \rceil,$$
$$m := (\lambda - 1)\Delta + 1.$$

Zu den Parametern z, τ sei ein Authentikationscode mit Fehlerwahrscheinlichkeit $\leq 2^{-\tau}$ für Nachrichten aus $\{0,1\}^z$ bestimmt.

S sei der Prozessor, der die Pseudosignatur zum Signieren verwendet. Für Phase 2 hat S als lokale Eingabe eine Nachricht $N \in \{0,1\}^z$.

Phase 1: Initialisieren einer Pseudosignatur

Protokoll 3.4 mit Parametern m, z, τ wird ausgeführt. Als Ergebnis erhält der Signierer S den **Signierschlüssel**

$$SSi = (SSi_1, \ldots, SSi_m),$$

wo jedes SSi_k eine Menge aus bis zu n Authentikationsschlüsseln ist, und jeder Prozessor P_i erhält einen privaten **Testschlüssel**

$$STe_i = (T_{i,1}, \ldots, T_{i,m}),$$

wo jedes $T_{i,k}$ ein Authentikationsschlüssel ist.

Phase 2: Signieren der Nachricht $N \in \{0,1\}^z$

Prozessor S bildet für $k = 1, \ldots, m$ die **Abschnitte**

$$\psi_k := \{\, Auth(T, N) \mid T \in SSi_k \,\}$$

und daraus die **Pseudosignatur**

$$\psi := (\psi_1, \ldots, \psi_m).$$

Phase 2 wird (für eine Phase 1) nur einmal ausgeführt.

Phase 3: Weitergeben einer Pseudosignatur

Hat ein Prozessor P_i ein Paar (N, ψ) in Phase 2 oder Phase 3 erhalten, so kann er dieses unverändert an einen oder mehrere Prozessoren senden.

Phase 4: Akzeptieren einer Pseudosignatur

Erhält ein Prozessor P_i ein Paar (N, ψ), so bestimmt er anhand seines Testschlüssels STe_i seine **Akzeptionsstufe**: Ist $\psi = (\psi_1, \ldots, \psi_m)$, so sei

$$a := \left|\left\{k \in \{1,\ldots,m\} \mid Auth(T_{i,k}, N) \in \psi_k\right\}\right|.$$

P_i ***k*-akzeptiert** das Paar, $k \in \{1, \ldots, \lambda\}$, genau dann, wenn gilt

$$(\lambda - k)\Delta < a.$$

P_i **lehnt** das Paar **ab**, wenn er es nicht einmal λ-akzeptieren kann.

Gemäß Definition enthält ψ auch Minisignaturen für S selbst. Aus Effizienzgründen kann man diese Minisignaturen hier (und entsprechend in Protokoll 3.4) weglassen. Dies hätte allerdings die Darstellung erschwert und an den Aufwandsaussagen in der hier betrachteten Genauigkeit nichts geändert.

3.1.5 Sicherheit gegen passive Angriffe

In diesem und dem nächsten Abschnitt wird die Sicherheit von Pseudosignaturen genauer betrachtet. Hier werden zunächst nur *passive Angriffe* betrachtet. Im Gegensatz zu einem aktiven Angriff können bei einem passiven die Störer nicht mit dem Angegriffenen interagieren.

Bezogen auf Pseudosignaturen können die Störer drei Ziele haben, entsprechend den drei Forderungen nach *Korrektheit*, *Unfälschbarkeit* und *Weitergebbarkeit*: Die Störer können versuchen,

- die Initialisierung so zu stören, daß eine *korrekte* Pseudosignatur später von wenigstens einem korrekten Prozessor nicht 1-akzeptiert wird,
- eine Pseudosignatur so zu *fälschen*, daß wenigstens ein korrekter Prozessor sie akzeptiert, oder
- eine echte Pseudosignatur so abzuändern, daß sie nicht mehr *weitergegeben* werden kann, d.h. für ein $k < \lambda$ zwar von einem korrekten Prozessor noch k-akzeptiert wird, von einem weiteren aber nicht mehr $(k+1)$-akzeptiert wird.

Das erste Problem taucht in gewisser Weise auch bei echten digitalen Signaturen auf: Die Erzeugung der Schlüssel (SSi, STe) geschieht dort zwar lokal durch den Signierer. Damit dessen Signatur später akzeptiert wird, muß STe allerdings zuverlässig an alle potentiellen Empfänger verteilt werden. Das erste Problem entspricht daher dem der Schlüsselverteilung.

Das Fälschungsproblem tritt bei echten digitalen Signaturen unverändert auf, das dritte Problem entspricht dort wiederum dem ersten.

Für die drei Probleme wird im folgenden Satz gezeigt, daß die Störer jeweils nur

mit exponentiell kleiner Wahrscheinlichkeit erfolgreich sind.

Satz 3.6: (Passive Sicherheit von Pseudosignaturen)
Die Phasen beziehen sich jeweils auf Schema 3.5.

a) (1-Akzeption von originalen Pseudosignaturen)
Mit Wahrscheinlichkeit $\geq 1-mn^4 2^{-\tau+1}$ gilt: Ist S korrekt und erhält (N, ψ) in Phase 2, so werden alle korrekte Prozessoren (N, ψ) 1-akzeptieren.

b) (Unfälschbarkeit von Pseudosignaturen)
Sei S korrekt. Dann ist für jedes Paar (N, ψ) die Wahrscheinlichkeit, daß es in Phase 4 ein bestimmter korrekter Prozessor λ-akzeptiert, $\leq mn2^{-\tau}$. Dies gilt selbst dann, wenn bereits die korrekte Minisignatur (N', ψ') unter ein $N' \neq N$ gegeben ist (d. h. als bedingte Wahrscheinlichkeit).

c) (Weitergebbarkeit von Pseudosignaturen)
Seien P und Q zwei korrekte Prozessoren, und sei $k < \lambda$ eine Akzeptionsstufe. Dann ist für jedes Paar (N, ψ) die Wahrscheinlichkeit, daß P es k-akzeptiert, Q hingegen nicht einmal $(k+1)$-akzeptiert, höchstens $2^{-\kappa}$.

d) Alle Phasen von Schema 3.5 verursachen einen in z, τ, κ und n polynomialen Aufwand. Es ist $m \leq 2\lambda^2\kappa$, und die Länge einer Pseudosignatur ist $\leq 2n\lambda^2\kappa(\tau+1)$ bit. ♦

Durch geeignete Wahl von τ werden alle drei Fehlerwahrscheinlichkeiten $\leq 2^{-\kappa}$: Zu betrachten ist nur (a), denn der Fehler von (b) ist kleiner als der von (a). Es ist mit $m \leq 2\lambda^2\kappa$

$$mn^4 2^{-\tau+1} \leq \lambda^2\kappa n^4 2^{-\tau+2} \leq 2^{-\kappa} \Leftrightarrow \tau \geq \kappa + 4\log_2(n) + \log_2(\kappa) + 2\log_2(\lambda) + 2.$$

Um τ nicht unnötig groß zu wählen, werden im folgenden aber beide Sicherheitsparameter τ und κ getrennt betrachtet.

Beweis: Ab Phase 2 sind die korrekten Prozessoren in Schema 3.5 deterministisch. Die Wahrscheinlichkeiten in Satz 3.6 sind daher nur durch das probabilistische Verhalten der Prozessoren in Phase 1 bestimmt. (Das probabilistische Verhalten der Störer nach Phase 1 spielt keine Rolle: für (a) ist dies offensichtlich, und (c) und (b) gelten jeweils für *alle* Paare (N, ψ) und ggf. (N', ψ').)

a) Nach Satz 3.5a1 hat S in Phase 1 mit Wahrscheinlichkeit $\geq 1-mn^4 2^{-\tau+1}$ von jedem korrekten Prozessor P_i alle Authentikationsschlüssel in STe_i erhalten. Folglich findet P_i in allen m Abschnitten eine Minisignatur für ihn und 1-akzeptiert.

b) Nach Abschnitt 3.1.2 ist die Wahrscheinlichkeit, die korrekte Authentikation für N zu einem unbekannten Authentikationsschlüssel zu finden, $\leq 2^{-\tau}$, selbst wenn bereits die Authentikation mit demselben Authentikationsschlüssel zu einem $N' \neq N$ bekannt ist. Eine Fälschung (N, ψ) ist erfolgreich, wenn P in wenigstens einem der m Abschnitte von ψ unter den $\leq n$ den Abschnitt bildenden Minisignaturen eine korrekte Authentikation für N findet. Die

Wahrscheinlichkeit hierfür ist $\leq mn2^{-\tau}$.

c) Es wird der schlimmste Fall betrachtet: Alle Prozessoren außer P und Q sind fehlerhaft und arbeiten zusammen, und die Störer kennen das Ergebnis von Phase 1, d.h. den Signierschlüssel SSi. Sie wissen daher zu jedem der m Abschnitte der Pseudosignatur, was die Minisignaturen für P und Q sind (falls vorhanden), nicht aber, welche für wen ist.

Enthält ein Abschnitt von SSi jeweils von P und Q einen Authentikationsschlüssel, so können die Störer entscheiden, ob sie beide Minisignaturen in ihre manipulierte Pseudosignatur ψ aufnehmen sollen, oder beide weglassen, oder je eine. Als schlimmster Fall wird angenommen, daß sie diese Wahlmöglichkeiten für jeden der m Abschnitte haben. Aus Symmetriegründen kann diese Entscheidung dann durch nur zwei Parameter beschrieben werden:

- g sei die Anzahl der Teile, für die beide Minisignaturen in ψ fehlen (*ganz* zerstört),
- h die Anzahl der Teile, für die genau eine Minisignatur fehlt (*halb* zerstört).

Zur Abkürzung sei

- x := $(k–1)\Delta$, d.h. die maximale Anzahl von P's Minisignaturen, die zerstört sein dürfen, damit P k-akzeptiert.

Sei B_h die Verteilungsfunktion der Binomialverteilung mit Parameter h und Wahrscheinlichkeit $1/2$ [ErSp_74]: Für $0 \leq k \leq h$ sei

$$B_h(k) := \sum_{z \leq k} \binom{h}{z} 2^{-h}.$$

Die Wahrscheinlichkeit, daß von den h Minisignaturen aus den Abschnitten, wo nur eine zerstört wurde, genau y zu P gehören, ist für $0 \leq y \leq h$

$$\binom{h}{y} 2^{-h}.$$

Die Störer haben Erfolg, wenn sie höchstens x Minisignaturen von P und mehr als $x + \Delta$ von Q zerstört haben, d.h. wenn gilt

$$\big(g + y \leq x\big) \wedge \big(g + (h - y) > x + \Delta\big). \tag{3.2}$$

Dies heißt $y \leq \min\{x - g, g + h - x - \Delta - 1\}$. Die Erfolgswahrscheinlichkeit für die Störer ist daher

$$\mathrm{P}(g, h) := B_h\big(\min\{x - g, g + h - x - \Delta - 1\}\big).$$

Es gilt

$$\min\{x - g, g + h - x - \Delta - 1\} \leq \frac{(x - g) + (g + h - x - \Delta - 1)}{2} = \frac{h}{2} - \frac{\Delta+1}{2}$$

$B_h(\bullet)$ steigt monoton, also ist

$$\mathrm{P}(g, h) \leq B_h\Big(\frac{h}{2} - \frac{\Delta+1}{2}\Big). \tag{3.3}$$

Nach [ErSp_74: Formel (3.8)] ist für jedes $\gamma \geq 0$

$$\sum_{z > h/2+\gamma} \binom{h}{z} 2^{-h} < \exp(-\frac{2\gamma^2}{h}).$$

Wegen der Symmetrie der Binomialverteilung gilt damit auch

$$B_h(\frac{h}{2} - \gamma - \frac{1}{2}) \leq \sum_{z < h/2-\gamma} \binom{h}{z} 2^{-h} < \exp(-\frac{2\gamma^2}{h}).$$

Dies wird nun auf (3.3) angewandt. Ziel ist $P(g, h) \leq 2^{-\kappa}$.

$$P(g, h) \leq B_h(\frac{h}{2} - \frac{\Delta+1}{2}) = B_h(\frac{h}{2} - \frac{\Delta}{2} - \frac{1}{2}) < \exp(-\frac{\Delta^2}{2h}) \leq 2^{-\kappa} = \exp(-\kappa \ln 2)$$

$$\Leftrightarrow \Delta^2 \geq 2h \ln 2 \ \kappa$$

Sei $\kappa^* := \ln 2 \ \kappa$. Dann ist also gesucht

$$\Delta^2 \geq 2h \ \kappa^*$$

Sicher ist $m \geq h$, d. h. es ist hinreichend, daß gilt

$$\Delta^2 \geq 2m \ \kappa^* = 2 \ ((\lambda-1)\Delta + 1) \ \kappa^*$$

$$\Leftrightarrow \quad \Delta^2 - 2\Delta(\lambda-1)\kappa^* - 2\kappa^* \geq 0$$

$$\Leftarrow \quad \Delta \geq (\lambda - 1)\kappa^* + \sqrt{(\lambda - 1)^2\kappa^{*2} + 2\kappa^*}$$

$$\Leftarrow \quad \Delta \geq 2(\lambda - 1)\kappa^* + 2\kappa^* = 2\lambda\kappa^*$$

$$\Leftrightarrow \quad \Delta \geq 2 \ln 2 \ \lambda\kappa.$$

d) Zum Aufwand der Phase 1 vgl. Satz 3.5d, und von Phase 2 und 4 vgl. Abschnitt 3.1.2. Der Rest ist offensichtlich.

Zur Länge von m und ψ: Mit der Festlegung für m gilt (mit $\kappa \geq 1$)

$$m \leq (\lambda-1) \ (2 \ln(2)\lambda\kappa + 1) + 1 \leq 2 \ln(2)\lambda^2\kappa + \lambda - 2 \ln(2)\lambda.$$

Die rechte Seite ist $\leq 2\lambda^2\kappa$:

$$2 \ln(2)\lambda^2\kappa + \lambda - 2 \ln(2)\lambda \leq 2\lambda^2\kappa \ \Leftarrow \ \lambda k \geq 1$$

Eine Pseudosignatur besteht dann aus $m \leq 2\lambda^2\kappa$ Abschnitten, und jeder Abschnitt aus n Authentikatoren der Länge $(\tau+1)$ bit, insgesamt also $2n\lambda^2\kappa(\tau+1)$ bit. ❐

3.1.6 Sicherheit gegen aktive Angriffe

Im folgenden werden die drei Probleme aus Abschnitt 3.1.5 unter dem Gesichtspunkt *aktiver* Angriffe betrachtet.

In den Abschnitten 3.1.6.2 bis 3.1.6.4 werden Aussagen bewiesen, die denen in Satz 3.6a,b,c entsprechen. Sie gelten für jedes Protokoll, das gewissen plausiblen Annahmen genügt (vgl. Abschnitt 3.1.6.1) und zeigen, daß eine große Anzahl authentisierter Protokolle so mit Pseudosignaturen implementiert werden kann, daß die Fehlerwahrscheinlichkeit exponentiell klein ist.

(In Abschnitt 3.2 wird dies für ein spezielles Verteilungsprotokoll genau

ausgeführt. Man sieht dort auch, daß eine spezielle Abschätzung der Fehlerwahrscheinlichkeiten zu erheblich kleineren Sicherheitsparametern führen kann.)

3.1.6.1 Ordentliche Protokolle

Authentisierte Protokolle, z. B. das in Abschnitt 3.2 verwendete Verteilungsprotokoll, sind im allgemeinen mit *jedem* hinreichend sicheren Authentikationsmechanismus implementierbar, d. h. ihre konkrete Ablaufstruktur hängt nicht vom verwendeten Authentikationsmechanismus oder dessen Sicherheitsparametern ab. Dies gilt insbesondere für ihre Rundenanzahl und die Gesamtzahl der Nachrichten, die ein Prozessor *authentisieren* oder *testen* muß.

Die meisten authentisierten Protokolle terminieren nach deterministisch polynomial vielen Runden, oder können zumindest nach deterministisch polynomial vielen Runden mit einer dann exponentiell kleinen Fehlerwahrscheinlichkeit abgebrochen werden. Ohne Beachtung des Aufwands für den Authentikationsmechanismus sind sie dann polynomial in n und möglicherweise einem Sicherheitsparameter ω.

Terminiert ein Protokoll mit obigen Eigenschaften in deterministisch R Runden, so kann man es, ohne Beachtung der Fehlerwahrscheinlichkeiten, mit Pseudosignaturen mit $\lambda = R$ Akzeptionsstufen implementieren:

In *einer* Runde kann eine Pseudosignatur höchstens *einmal* sequentiell weitergegeben werden (aber natürlich mehrmals parallel), in R Runden also höchstens R mal. Ein Prozessor darf daher in der r-ten Runde nur solche Pseudosignaturen akzeptieren, die er wenigstens r-akzeptiert.

Im folgenden wird gezeigt, daß man die Sicherheitsparameter κ, τ des Pseudosignaturschemas polynomial in n und einem Sicherheitsparameter ω so wählen kann, daß die Eigenschaften der *Korrektheit*, *Unfälschbarkeit* und *λ-Weitergebbarkeit* jeweils mit Wahrscheinlichkeit $\geq 1–2^{-\omega}$ erfüllt sind.

Hierzu wird die Klasse der betrachteten Protokolle etwas formaler eingeschränkt:

Sei ω ein Sicherheitsparameter. Ein Protokoll, das ein Pseudosignaturschema verwendet, heiße **ordentlich**, wenn gilt:

- Es gibt ein Polynom $g(n,\omega)$, so daß im Protokoll Phase 1 von Schema 3.5 höchstens $g(n,\omega)$ mal ausgeführt wird.
- Es zerfällt in zwei Phasen:
  - In der ersten Phase werden alle $\leq g(n,\omega)$ später benötigten Pseudosignaturen initialisiert (Schema 3.5, Phase 1). Sonst geschieht hier nichts.
  - In der zweiten Phase wird auf das Ergebnis der ersten Phase nur noch mit Schema 3.5 zugegriffen.

    Dies heißt insbesondere: Ein korrekter Prozessor P_i greift auf STe_i nur noch zu, indem er Phase 4 von Schema 3.5 ausführt. Die Störer

können daher keine Information über STe_i erhalten, außer indem sie P_i dazu bewegen, Pseudosignaturen zu testen.

- Es gibt ein Polynom $h(n,\omega)$, so daß jeder korrekte Prozessor in jedem Protokollablauf Phase 4 von Schema 3.5 bezogen auf eine bestimmte Phase 1 von Schema 3.5 höchstens $h(n,\omega)$ mal ausführt.
- Die Parameter κ und τ des Pseudosignaturschemas sind beliebige Protokollparameter. λ und z sind polynomial in n und ω und unabhängig von κ und τ.

3.1.6.2 Korrektheit

Aktive Angriffe sind bei Pseudosignaturen für das erste in Abschnitt 3.1.5 genannte Problem, die 1-Akzeption von originalen Pseudosignaturen, bedeutungslos: In Satz 3.6a wurde gezeigt, daß das Initialisierungsprotokoll trotz beliebigen Störerverhaltens fast sicher eine korrekte Initialisierung findet. Es folgt daher unmittelbar der folgende Satz:

Satz 3.7: (Korrektheit von Pseudosignaturen)
Wird für ein ordentliches Protokoll ein Parameter

$$\tau \geq \omega + \log_2(\kappa) + 2 + \log_2(g(n,\omega)\lambda^2 n^4)$$

gewählt, so gilt mit Wahrscheinlichkeit $\geq 1–2^{-\omega}$:

Erhält ein korrekter Prozessor von einem korrekten Signierer eine Pseudosignatur, so 1-akzeptiert er diese. ♦

Beweis: Nach Satz 3.6a gilt dies mit Wahrscheinlichkeit $\geq 1–mn^4 2^{-\tau+1}$ für jede der $\leq g(n,\omega)$ Pseudosignaturen des Protokolls. Also ist der Fehler insgesamt, mit $m \leq 2\lambda^2\kappa$, höchstens $g(n,\omega)\lambda^2\kappa n^4 2^{-\tau+2}$, und es ist

$$g(n,\omega)\lambda^2\kappa n^4 2^{-\tau+2} \leq 2^{-\omega} \Leftrightarrow \tau \geq \omega + \log_2(\kappa) + 2 + \log_2(g(n,\omega)\lambda^2 n^4).$$ ❒

3.1.6.3 Aktive Angriffe auf die Unfälschbarkeit

Satz 3.8: (Unfälschbarkeit von Pseudosignaturen)
Wird für ein ordentliches Protokoll ein Parameter

$$\tau \geq \omega + \log_2(\kappa) + 1 + \log_2(g(n,\omega)\, h(n,\omega)\, \lambda^2 n^2)$$

gewählt, so gilt mit Wahrscheinlichkeit $\geq 1–2^{-\omega}$.

Kein korrekter Prozessor λ-akzeptiert eine gefälschte Pseudosignatur. ♦

Beweis: Zunächst betrachte man Schema 3.5 für eine der $\leq g(n,\omega)$ Ausführungen von Phase 1, d.h. für eine Pseudosignatur, und einen der $\leq n$ korrekten Prozessoren P_i.

Da das Protokoll ordentlich ist, wird P_i höchstens $h(n,\omega)$ Werte als Pseudosignatur mit Phase 4 von Schema 3.5 testen. Die Störer sind bezüglich P_i dann er-

folgreich, wenn P_i dabei wenigstens einmal einen gefälschten Wert λ-akzeptiert.

Als schlimmsten Fall nehme man an, die Störer kennen bereits die korrekte Pseudosignatur (N', ψ') und sogar für jeden Authentikationsschlüssel $T \in STe_i$ die Authentikation $Auth(T, N')$.

Im für die Störer günstigsten Fall können sie an P_i sequentiell $h(n,\omega)$ Paare (N, ψ) ausprobieren, und sie erfahren, bevor sie das $(k+1)$-te Paar wählen, ob P_i das k-te Paar wenigstens λ-akzeptiert. Sie sind erfolgreich, wenn P_i eines der $h(n,\omega)$ Paare wenigstens λ-akzeptiert.

O. B. d. A. kann man annehmen, das Störprotokoll sei deterministisch: Nachdem die Störer das Ergebnis des k-ten Paars erhalten haben, haben sie im allgemeinen mehrere, aber nur endlich viele Möglichkeiten für das $(k+1)$-te Paar. Darunter gibt es wenigstens eines, das ihre Erfolgswahrscheinlichkeit maximiert, und offensichtlich wird die Erfolgswahrscheinlichkeit der Störer höchstens größer, wenn sie *deterministisch* ein solches wählen.

Damit kann man die Erfolgswahrscheinlichkeit des aktiven Angriffs abschätzen, indem man annimmt, die Störer könnten *alle* $h(n,\omega)$ Paare an P_i ausprobieren. Sie ist daher mit Satz 3.6b,d $\leq h(n,\omega)\, mn2^{-\tau} \leq h(n,\omega)\lambda^2\kappa n2^{-\tau+1}$.

Für alle n Prozessoren P_i und alle $g(n,\omega)$ Pseudosignaturen erhält man damit eine Fehlerwahrscheinlichkeit

$$\leq g(n,\omega)\, h(n,\omega)\, \lambda^2\kappa n^2 2^{-\tau+1} \leq 2^{-\omega}$$

$$\Leftrightarrow \tau \geq \omega + \log_2(\kappa) + 1 + \log_2(g(n,\omega)\, h(n,\omega)\, \lambda^2 n^2).$$ ❐

3.1.6.4 Aktive Angriffe auf die Weitergebbarkeit

Für das dritte Problem aus Abschnitt 3.1.5, die Weitergabemöglichkeit von Pseudosignaturen, können aktive Angriffe genauso wie für das zweite Problem beschrieben werden, d. h. die Störer können statt einer Pseudosignatur mehrere an P und Q ausprobieren.

Das einfachste Beispiel für einen solchen Angriff ist, daß ein Störer aus ψ genau eine Minisignatur wegläßt. 1-akzeptiert P das so manipulierte ψ nicht mehr, so war diese Minisignatur für ihn, andernfalls nicht. Kann dieser Angriff häufig genug iteriert werden, so können alle für P bestimmten Minisignaturen in ψ bestimmt werden, und es kann dann z. B. eine „Pseudosignatur“ erzeugt werden, die P 1-akzeptiert, aber alle anderen Prozessoren zurückweisen.

Im allgemeinen Fall können P und Q auf einen Versuch der Störer auf mehr als eine Art reagieren. Die Eigenschaft des ordentlichen Protokolls, auf STe nur mittels Phase 4 von Schema 3.5 zuzugreifen, garantiert aber, daß jeder Prozessor auf einen Versuch nur auf $\lambda+1$ wesentlich verschiedene Arten reagieren kann; für jedes mögliche Ergebnis von Phase 4 auf eine. Zur Abschätzung der Fehlerwahrscheinlichkeit wird bestimmt, wieviele verschiedene Versuche die Störer insgesamt durchführen könnten, d. h. für *alle* möglichen alternativen

Reaktionen von P und Q zusammen. Ist diese Zahl $\leq C$, so ist die Fehlerwahrscheinlichkeit des aktiven Angriffs $\leq C\, 2^{-\kappa}$.

Dies führt zu folgendem Satz 3.9.

Satz 3.9: (Weitergebbarkeit von Pseudosignaturen)

Wird für ein ordentliches Protokoll der Parameter

$$\kappa \geq \omega + (2\, h(n,\omega) + 1)\, \log_2(\lambda+1) + \log_2(n^2\, g(n,\omega))$$

gewählt, so gilt mit Wahrscheinlichkeit $\geq 1-2^{-\omega}$:

Wird eine Pseudosignatur von einem korrekten Prozessor k-akzeptiert für ein $k < \lambda$, so würde sie jeder andere korrekte Prozessor $(k+1)$-akzeptieren. ♦

Beweis: Man betrachtet zunächst nur zwei korrekte Prozessoren P und Q, eine Ausführung von Phase 1 von Schema 3.5 und eine Akzeptionsstufe $k < \lambda$. Das Ziel der Störer ist, daß im Protokoll P oder Q ein Paar (N, ψ) k-akzeptiert, das Q bzw. P nicht $(k+1)$-akzeptieren würde. Ein solches Paar heiße **falsch *k*-akzeptiert**.

Es wird der schlimmste Fall angenommen: Alle Prozessoren außer P und Q sind Störer, und die Störer kennen auch den korrekten Signierschlüssel SSi (etwa weil der Signierer ein Störer ist). Unbekannt ist ihnen lediglich die Zuordnung Z der Authentikationsschlüssel von P und Q zu diesen beiden Prozessoren.

In einem ordentlichen Protokoll haben die Störer vor Beginn von Phase 2 keine Information über die Zuordnung Z (Satz 3.5b). In Phase 2 ist die einzige Möglichkeit, Information über Z zu erhalten, P und Q dazu zu veranlassen, Phase 4 von Schema 3.5 auszuführen und das Resultat bekanntzugeben. Phase 4 hat $(\lambda+1)$ mögliche Resultate (λ Akzeptionsstufen und „abgelehnt“).

Im für die Störer günstigsten Fall können sie an P und Q jeweils sequentiell $h(n,\omega)$ Paare (N, ψ) ausprobieren, und sie erhalten, bevor sie das i-te Paar ausprobieren, das Ergebnis von Phase 4 für das $(i-1)$-te Paar.[33] Sie sind erfolgreich, wenn P eines der $h(n,\omega)$ Paare falsch k-akzeptiert.

Um die Erfolgswahrscheinlichkeit eines solchen Angriffs abzuschätzen, werden o. B. d. A. nur deterministische Störprotokolle betrachtet:

In einem konkreten Angriff sei $\alpha_i = (N_i, \psi_i, X_i)$ das i-te Paar, das die Störer an $X_i \in \{P, Q\}$ ausprobieren, und β_i sei die dazugehörende Reaktion von X_i.

Nachdem die Störer β_i erhalten haben, haben sie im allgemeinen mehrere Möglichkeiten, α_{i+1} zu bestimmen. Da es nur endlich viele mögliche Pseudosignaturen gibt, sind dies jedoch endlich viele, und für jede davon kann im Prinzip die Wahrscheinlichkeit bestimmt werden, daß der Angriff erfolgreich

---

[33] Werden Pseudosignaturen, wie in Abschnitt 3.1.6.1 beschrieben, so verwendet, daß jeder Prozessor weiß, wie er eine erhaltene Pseudosignatur akzeptieren muß, so erhalten die Störer nicht eine aus $\lambda+1$ möglichen Reaktionen, sondern nur eine aus zwei, nämlich entweder „abgelehnt“ oder „passend akzeptiert“.

enden wird, unter der Bedingung, daß er bisher so verlief, wie er verlief. Offensichtlich wird die Erfolgswahrscheinlichkeit der Störer höchstens größer, wenn sie *deterministisch* jeweils einen Wert α_{i+1} wählen, der ihre Erfolgswahrscheinlichkeit maximiert.

Ein deterministisches Störprotokoll wird durch einen **Strategiebaum** beschrieben:

Ein solcher ist ein Baum der Tiefe $2h(n,\omega)-1$. Jeder Knoten enthält einen Versuch der Störer, d.h. ein Tripel $\alpha_i = (N_i, \psi_i, X_i)$. Die Wurzel enthält den ersten Versuch α_0. Jeder innere Knoten mit Inhalt α_i hat $\lambda+1$ Kinder, für jedes mögliche β_i eines (also eines für jede der $(\lambda+1)$ möglichen Ausgaben von Phase 4 von Schema 3.5, ausgeführt von X_i für (N_i, ψ_i)). Die Kinder von α_i entsprechen den deterministischen Reaktionen des Störprotokolls auf die $\lambda+1$ möglichen Reaktionen β_i.

Ein Strategiebaum hat, wie man leicht nachrechnet, $\leq \lambda^{-1}(\lambda+1)^{2h(n,\omega)+1}$ Knoten und enthält daher ebensoviele Pseudosignaturen. Die Erfolgswahrscheinlichkeit der Störer wird nun einfach dadurch abgeschätzt, daß angenommen wird, die Störer könnten alle diese Pseudosignaturen an P oder Q ausprobieren. Sie ist daher nach Satz 3.6c

$$\leq \lambda^{-1}(\lambda+1)^{2h(n,\omega)+1}\, 2^{-\kappa}.$$

Betrachtet man die Falschakzeption nicht nur für ein Paar $\{P, Q\}$, eine Akzeptionsstufe $k < \lambda$ und eine Pseudosignatur, sondern für alle $\leq n^2/2$ Paare korrekter Prozessoren, alle λ Akzeptionsstufen und alle $\leq g(n,\omega)$ Ausführungen von Schema 3.5, Phase 1, so erhöht sich die Fehlerwahrscheinlichkeit auf

$$\leq n^2\, g(n,\omega)\, (\lambda+1)^{2h(n,\omega)+1}\, 2^{-\kappa},$$

und dies ist

$$\leq 2^{-\omega} \Leftrightarrow \kappa \geq \omega + (2\, h(n,\omega) + 1)\, \log_2(\lambda+1) + \log_2(n^2\, g(n,\omega)).$$ ❐

Die Festlegungen für τ und κ in Satz 3.7 bis Satz 3.9 sind zusammen polynomial in n und ω, d.h. es ist gezeigt, daß man ein ordentliches Protokoll so mit Pseudosignaturen implementieren kann, daß die Fehlerwahrscheinlichkeit exponentiell klein ist.

3.2 Byzantinische Verteilung mit Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ in t+1 Runden

In Abschnitt 3.2.1 wird zunächst ein bestimmtes authentisiertes Verteilungsprotokoll betrachtet, in dem Pseudosignaturen verwendet werden können. In Abschnitt 3.2.2 wird gezeigt, daß dieses das byzantinische Verteilungsproblem mit exponentiell kleiner Fehlerwahrscheinlichkeit löst, und was die möglichen aktiven Angriffe für die Wahl von κ, τ genau bedeuten. In Abschnitt 3.2.3 wird schließlich beschrieben, wie mit einem in u logarithmischen Vorbereitungsprotokoll u Verteilungen durchgeführt werden können.

3.2.1 Modifiziertes DOLEV/STRONG-Protokoll

Das folgende Verteilungsprotokoll ist eine Modifikation des authentisierten Verteilungsprotokolls von DOLEV und STRONG [DoSt_83: Th. 3]. Unter anderem war überall festzulegen, welche Akzeptionsstufe verlangt wird, und die Anzahl möglicher aktiver Angriffe (vgl. Abschnitt 3.1.6) wurde so gering wie möglich gehalten.

Für das Protokoll wird ein **Sicherheitsparameter** σ festgelegt, so daß eine Fehlerwahrscheinlichkeit von $2^{-\sigma}$ akzeptabel ist.

Im Protokoll werden Pseudosignaturen mit $\lambda = t+1$ Akzeptionsstufen verwendet. Die Nachrichtenlänge z ist ein Protokollparameter, die Sicherheitsparameter τ und κ für Schema 3.5 ergeben sich aus σ (vgl. Protokoll).

Für jede geplante Protokollausführung werden im Vorbereitungsprotokoll für den Transmitter T eine, für jeden anderen Prozessor zwei Pseudosignaturen initialisiert. Die erste bzw. zweite Pseudosignatur wird **A-** bzw. **B-Signatur** genannt.

Damit zu einer Pseudosignatur stets klar ist, auf welche der $2n-1$ Ausführungen von Schema 3.5 sie sich bezieht, treten im Protokoll Pseudosignaturen nur in Tripeln (i, α, ψ) auf, wobei ψ die eigentliche Pseudosignatur, $\alpha \in \{A,B\}$ den „Typ“ und P_i den (angeblichen) Signierer bezeichnet.

Ein solches Tripel wird von einem Prozessor als Pseudosignatur unter v ***k*-akzeptiert**, wenn der Prozessor (v, ψ) bezüglich des durch (i, α) bezeichneten Testschlüssels k-akzeptiert.

Protokollnachrichten enthalten jeweils einen Wert v und eine Menge von bis zu t+1 Tripeln (i, α, ψ). Für jede Akzeptionsstufe $k \in \{1, \ldots, t+1\}$ heiße eine Protokollnachricht N ***k*-konsistent** aus der Sicht von P_i, wenn gilt: Es ist

$$N = \Big(v, \big\{(i_1, \alpha_1, \psi_1), \ldots, (i_k, \alpha_k, \psi_k)\big\}\Big),$$

wobei gilt:

- Für ein $j \in \{1, \ldots, k\}$ ist $P_{i_j} = T$ und $\alpha_{i_j} = A$, d.h. N enthält eine A-Signatur des Transmitters.
- Alle i_j sind paarweise verschieden und $i \notin \{i_1, \ldots, i_j\}$, d.h. N enthält k Pseudosignaturen von k verschiedenen Prozessoren, nicht jedoch von P_i selbst.
- P_i k-akzeptiert alle Tripel (i_j, α_j, ψ_j) als Pseudosignaturen unter v.

Im Protokoll muß jeder Prozessor P maximal zwei aus seiner Sicht passend k-konsistente Protokollnachrichten **weiterleiten**. P leitet eine Protokollnachricht N weiter, indem

- er den in N enthaltenen Wert v pseudosigniert, wobei er beim ersten mal seine A-, beim zweiten mal seine B-Signatur verwendet, seine Pseudosignatur der in N enthaltenen Menge hinzufügt, und
- die so erweiterte Nachricht an alle Prozessoren sendet, von denen noch keine Pseudosignatur unter v in N enthalten ist.

In [DoSt_83] sind in den k-konsistenten Protokollnachrichten die Signaturen der Prozessoren nicht wie oben nebeneinander angeordnet, sondern hintereinander, d.h. der obigen Protokollnachricht N entspricht dort die Nachricht

$$Sign_{i_1}(Sign_{i_2}(\ldots Sign_{i_k}(v)\ldots)),$$

falls $T = P_{i_k}$ ist. Wie Satz 3.10 zeigt, wird durch das Nebeneinanderstellen die Fehlertoleranz des Protokolls jedoch nicht beeinträchtigt, und die Pseudosignaturen werden etwas kürzer.

Die Menge AKZ_i in Protokoll 3.6 enthält die Werte, die P_i vom Transmitter akzeptiert. Enthält am Ende AKZ_i genau einen Wert, so nimmt P_i diesen als Endwert. Ansonsten weiß P_i, daß der Transmitter fehlerhaft ist, und (wie zu zeigen ist) jeder korrekte Prozessor entscheidet sich für „Transmitterfehler“.

Die Mengen ALT_i garantieren, daß jeder korrekte Prozessor P_i von jedem anderen Prozessor P_x maximal zwei Protokollnachrichten auf Konsistenz prüft. Hierdurch wird die Information, die P_i über seine Minisignaturen preisgibt, im Sinne von Abschnitt 3.1.6.4 beschränkt.

Protokoll 3.6: Byzantinische Verteilung für $t < n$
mit Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ in t+1 Runden
mit Parametern t, z, σ

Parameter t gibt die maximal zu tolerierende Anzahl an Störern an, z die Länge des zu verteilenden Wertes. Zum Sicherheitsparameter σ seien

$$\kappa := \lceil \sigma + 18 \log_2(n) + \log_2(72) \rceil$$

$$\tau := \lceil \sigma + \log_2(\kappa) + 7 \log_2(n) + \log_2(24) \rceil.$$

Phase 1: Vorbereitungsprotokoll

Mit Schema 3.5, Phase 1, mit $\lambda = t$+1 und Parametern z, τ und κ werden $2n$–1 Pseudosignaturen initialisiert. Dabei tritt T einmal, jeder andere Prozessor zweimal als Signierer auf.

Die $2n$–1 Pseudosignaturen werden wie oben erläutert bezeichnet.

Phase 2: Verteilungsprotokoll

Der Transmitter T hat als lokale Eingabe seinen Anfangswert $v \in \{0,1\}^z$.

[0] Der Transmitter leitet seinen Anfangswert v weiter, d. h. er pseudosigniert ihn und sendet ihn an alle anderen Prozessoren.

Jeder Prozessor $P_i \neq T$ initialisiert zwei Mengen ALT_i und AKZ_i als „leer“.

Für $k = 1,\ldots,t$+1 und jeweils alle Prozessoren $P_i \neq T$:

[k] a. P_i bildet die Menge $N_{i,k}$ der **formal neuen** Protokollnachrichten:

Für jede Protokollnachricht N, die in Runde [k–1] an P_i gesendet wurde: Sei P_x der Absender von N. Wenn eine Pseudosignatur (x,α,ψ) in N enthalten ist und $(x, \alpha) \notin ALT_i$, so nimmt P_i die Nachricht N in $N_{i,k}$ auf und fügt das Paar (x, α) zu ALT_i hinzu; ansonsten wird N als „Quasi-wiederholung“ ignoriert.

b. P_i bildet die Menge $V_{i,k}$ aller ***k*-konsistenten** Nachrichten in $N_{i,k}$. $V_{i,k}$ sei lexikographisch geordnet.

P_i nimmt alle Werte, die in den Nachrichten in $V_{i,k}$ enthalten sind, in AKZ_i auf.

c. Ist $k \leq t$, $V_{i,k} \neq \emptyset$ und hat P_i bislang noch *keine* Protokollnachricht weitergeleitet, so leitet er die ersten beiden Protokollnachrichten in $V_{i,k}$ mit verschiedenen Werten weiter, bzw. die erste, falls alle Protokollnachrichten in $V_{i,k}$ denselben Wert enthalten.

d. Ist $k \le t$, $V_{i,k} \neq \emptyset$ und hat P_i bislang genau *eine* Protokollnachricht weitergeleitet, die einen Wert v' enthielt, so leitet er die erste Protokollnachricht aus $V_{i,k}$ weiter, die einen Wert $v'' \neq v'$ enthält, falls eine solche existiert.

Endwert:

Der Transmitter entscheidet sich für seinen eigenen Anfangswert.

Hat $P_i \neq T$ genau einen Wert v in AKZ_i aufgenommen, so entscheidet er sich für v. Sonst entscheidet er sich für „Transmitterfehler".

3.2.2 Beweis des byzantinischen Verteilungsprotokolls

Der Beweis, daß Protokoll 3.6 tatsächlich das tut, was es verspricht, erfolgt in zwei Schritten.

In Satz 3.10 wird zunächst bewiesen, daß das Protokoll das byzantinische Verteilungsproblem deterministisch lösen würde, wenn es mit einem fehlerfreien Pseudosignaturschema implementiert würde. Dies entspricht etwa dem Beweis in [DoSt_83], und auf entsprechende Art ist die Fehlertoleranz der meisten authentisierten Verteilungsprotokolle (oder ähnlicher Protokolle) bewiesen worden.

Der Fehler, den man mit der Annahme eines fehlerfreien Pseudosignaturschemas macht, wird dann in Satz 3.11 abgeschätzt. Er ist zugleich die Fehlerwahrscheinlichkeit des Protokolls.

Satz 3.10: (Deterministische Version der byzantinischen Verteilung)

a) Angenommen, für alle in einem Protokollablauf vorkommenden Pseudosignaturen ψ, alle $\alpha \in$ {A,B}, alle korrekten Prozessoren P und Q, alle Werte v und alle Runden [k] gilt:

A1. Ist ψ die α-Signatur von P unter den Wert v, so 1-akzeptiert Q dies ψ.

A2. Akzeptiert in Runde [k] Prozessor P den Wert ψ als eine von Q stammende α-Signatur unter den Wert v, so hat Q spätestens in Runde [k–1] eine Protokollnachricht weitergeleitet, die seine α-Signatur unter den Wert v enthielt.

A3. Wird ein Wert von P als α-Signatur ψ irgendeines Prozessors R k-akzeptiert, so würde Q diesen Wert (k+1)-akzeptieren.

Dann löst dieser Protokollablauf das byzantinische Verteilungsproblem für jedes $t < n$.

b) Der Rechenaufwand des Protokolls ist polynomial in n, z und σ.

Das Vorbereitungsprotokoll benötigt im besten Fall 3, im schlimmsten Fall $2n^2$ Runden. Sei $\gamma := \big(\log(z)\ (\sigma+\log(n))\ (\sigma+\log(n)+\log(\log(z)))\big)$. Dann müssen im schlimmsten Fall in $O(n^4)$ Nachrichten $O(n^8\gamma)$ bit über sichere,

nichtabhörbare Verbindungen und in $O(n^3)$ Nachrichten $O(n^8\gamma)$ bit über einen zuverlässigen Verteilkanal gesendet werden.

Das Verteilungsprotokoll benötigt stets t+1 Runden, in denen in weniger als $2n^2$ Nachrichten $O(n^2z + n^6(\sigma+\log(n))^2)$ bit über sichere, nicht notwendigerweise nichtabhörbare Verbindungen gesendet werden. ♦

Satz 3.11: (Byzantinische Verteilung)
Protokoll 3.6 löst für $t < n$ in jedem Protokollablauf mit Wahrscheinlichkeit $\geq 1–2^{-\sigma}$ das byzantinische Verteilungsproblem. ♦

Beweis von Satz 3.10:

a) Fall 1: Der Transmitter ist korrekt und leitet daher in Runde [0] seinen Wert v weiter. Jeder korrekte Prozessor P_i betrachtet in Runde [1] diese Protokollnachricht als 1-konsistent (A1) und nimmt v in AKZ_i auf. Da der Transmitter keine weiteren Nachrichten sendet, kann wegen (A2) im Protokoll keine k-konsistente Protokollnachricht auftauchen, die einen anderen Wert als v enthält, d.h. AKZ_i enthält am Ende nur den Wert v.

Fall 2: Der Transmitter ist fehlerhaft.

Zunächst wird gezeigt: Gilt für einen korrekten Prozessor P_i am Ende des Protokolls $v \in AKZ_i$, so gilt für jeden korrekten Prozessor P_j am Ende $v \in AKZ_j$, oder es gilt für jeden korrekten Prozessor P_j am Ende $|AKZ_j| \geq 2$.

Grund: Sei [k] eine Runde, in dem P_i eine k-konsistente Protokollnachricht N in seine Menge $V_{i,k}$ aufgenommen hat, die den Wert v enthält.

- Ist $k < t$+1, so ist zu unterscheiden, ob P_i die Protokollnachricht N weiterleitet oder nicht.
  - Leitet er sie weiter, so ist sie für jeden korrekten Prozessor P_j wegen (A1, A3) (k+1)-konsistent: Sie war für P_i k-konsistent, d.h. enthielt ausreichend viele und passende für P_i k-akzeptable Pseudosignaturen, und enthält nun zusätzlich eine Pseudosignatur von P_i, die für P_j sogar 1-akzeptabel ist. P_i verwendet seine A- und B-Signatur höchstens einmal, so daß N bei P_j den Filterschritt (a) in Schritt [k+1] übersteht. Folglich gilt danach $v \in AKZ_j$ für jeden korrekten P_j.
  - Ansonsten gibt es nach den Regeln in Phase 2, Runde [k], (c) und (d) zwei Gründe, weshalb P_i die Nachricht nicht weiterleitet:

    P_i hat den Wert v schon früher in einer passenden Protokollnachricht weitergeleitet, und es gilt mit derselben Begründung wie oben am Ende $v \in AKZ_j$ für jeden korrekten P_j.

    P_i hat bereits zwei Werte weitergeleitet. Dann hat jeder korrekte P_j mit derselben Begründung wie oben diese beiden Werte auch erhalten, und es gilt am Ende $|AKZ_j| \geq 2$ für jeden korrekten P_j.
- Ist $k = t$+1, so ist in N eine Pseudosignatur eines korrekten Prozessors P_h

enthalten, der v wegen (A2) daher in einer früheren Runde weitergeleitet und daher *erhalten* haben muß. Nach dem für $k < t+1$ Gesagten gilt daher $v \in AKZ_j$ für jeden korrekten P_j. ◊

Hieraus folgt: Gilt am Ende $v \in AKZ_i$ für einen korrekten Prozessor P_i, so gilt entweder $AKZ_j = \{v\}$ für *alle* korrekten Prozessoren P_j, und *alle* erhalten v als Endwert, oder es gilt $|AKZ_j| \geq 2$ für *alle* korrekten Prozessoren P_j, und *alle* entscheiden sich für „Transmitterfehler". Sonst gilt am Ende $AKZ_i = \emptyset$ für alle korrekten Prozessoren P_i, und *alle* entscheiden sich für „Transmitterfehler".

b) Nach Protokoll sind κ, τ beide = $O(\sigma+\log(n))$. Damit ist die Polynomialität des Rechenaufwands offensichtlich.

Zum Vorbereitungsprotokoll: Nach Satz 3.5d benötigt Phase 1 von Schema 3.5 im besten Fall 3, im schlimmsten $2n^2$ Runden, und alle Pseudosignaturen können parallel initialisiert werden.

In einer Runde werden, faßt man alle Nachrichten vom selben Absender an denselben Adressaten zu einer zusammen, höchstens $O(n^2)$ Nachrichten über Verbindungen und $O(n)$ Nachrichten über einen Verteilkanal gesendet. Hieraus ergeben sich die Nachrichtenanzahlen $O(n^4)$ bzw. $O(n^3)$.

Im Vorbereitungsprotokoll werden $O(n)$ Pseudosignaturen parallel initialisiert. Hierfür müssen parallel $O(n)$ mal die Phasen 1 bis 3 von Protokoll 3.4 ausgeführt werden, und dies im schlimmsten Fall $O(n^2)$ mal.

Gesucht ist also der Aufwand der Phasen 1 bis 3, und diesen muß man mit $O(n^3)$ multiplizieren: Mit $\xi := \log(z)\ (\tau + \log(\log(z))\ )$ sind nach Satz 3.5d in diesen Phasen $O(mn^3\xi)$ bit über sichere, nichtabhörbare Verbindungen zu senden und zu veröffentlichen.

Es ist nach Satz 3.6d $m = O(\lambda^2\kappa) = O(n^2(\sigma+\log(n)))$. Folglich ist

$$O(mn^3\xi) = O(n^5(\sigma+\log(n))\xi).$$

Mit $\gamma = (\sigma+\log(n))\xi$ und durch Multiplikation mit $O(n^3)$ erhält man die in Satz 3.10 angegeben Werte.

Zum Verteilungsprotokoll: Jeder Schritt [0] bis [t] benötigt eine eigene Runde. In Schritt [t+1] werden keine Nachrichten mehr gesendet. Das Verteilungsprotokoll benötigt daher t+1 Runden.

Der korrekte Transmitter sendet n–1 Nachrichten. Jeder andere korrekte Prozessor P_i leitet höchstens zwei Nachrichten weiter, und jede Weiterleitung bedeutet maximal n–2 Nachrichten. Insgesamt sind dies maximal

$$n-1 + 2\ (n-1)(n-2) < 2n^2$$

Nachrichten.

Jede Nachricht enthält einen Wert der Länge z bit und $\leq n$ Tripel (i, α, ψ). i ist ein Index aus $\{0, \ldots, n\}$ und hat daher die Länge $O(\log(n))$ bit, $\alpha \in \{A,B\}$ hat die Länge 1 bit, und ψ hat nach Satz 3.6d die Länge $\leq 2n(t+1)^2\kappa(\tau+1)$ bit $= O(n^3(\sigma+\log(n))^2)$ bit. Damit ist die Länge einer Nachricht $O(z + n^4(\sigma+\log(n))^2)$

bit.

Für alle $O(n^2)$ Nachrichten erhält man $O(n^2 z + n^6(\sigma+\log(n))^2)$ bit. ❐

Bevor Satz 3.11 bewiesen wird, soll zunächst kurz gezeigt werden, welche Resultate die Sätze 3.7 bis 3.9 liefern würden:

Das obige Protokoll ist offensichtlich *ordentlich*. Im Protokoll werden $2n-1$ Pseudosignaturen verwendet, d. h. es ist

$$g(n,\sigma) = 2n-1.$$

Die Menge ALT_i und der „Filterschritt" (a) in Protokoll 3.6, Phase 2, Schritt $[k]$, garantieren, daß ein korrekter Prozessor von jedem Prozessor außer dem Transmitter höchstens zwei, vom Transmitter höchstens eine Protokollnachricht als formal neu betrachtet, von allen Prozessoren zusammen also $\leq 2n-3$. Jede Protokollnachricht kann eine bestimmte Pseudosignatur höchstens einmal enthalten, woraus folgt

$$h(n,\sigma) \leq 2n-3.$$

Damit ist für eine Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ *jeweils* hinreichend (d. h. alle zusammen haben dann eine Fehlerwahrscheinlichkeit $\leq 3 \cdot 2^{-\sigma}$):

$$\tau \geq \sigma + \log_2(\kappa) + 3 + 7\log_2(n)$$

$$\kappa \geq \sigma + (4n-2)\log_2(n+1) + 1.$$

Der Wert von κ wächst mit $n \log(n)$. Der Nachweis, daß der nur mit $\log(n)$ wachsende Wert für κ in Protokoll 3.6 genügt, wird durch eine genauere Betrachtung des Strategiebaums der Störer für Protokoll 3.6 erbracht.

Man sieht aber an der groben Abschätzung schon, daß mit Protokoll 3.6 eine exponentiell kleine Fehlerwahrscheinlichkeit erreicht werden kann.

Beweis von Satz 3.11: Mit Satz 3.10 genügt es zu zeigen, daß für alle Fehlersituationen die Voraussetzungen A1 bis A3 von Satz 3.10 mit Wahrscheinlichkeit $W \leq 2^{-\sigma}$ nicht zutreffen.

Sei W_i die Wahrscheinlichkeit, daß (Ai) nicht erfüllt ist. W wird abgeschätzt durch

$$W \leq W_1 + W_2 + W_3.$$

W_1 und W_2 werden wie in Satz 3.7 bzw. Satz 3.8 abgeschätzt:

$$W_1 \leq g(n,\sigma)\lambda^2\kappa n^4 2^{-\tau+2} \leq \kappa n^7 2^{-\tau+3}.$$

$$W_2 \leq g(n,\sigma)\, h(n,\sigma)\, \lambda^2\kappa n^2 2^{-\tau+1} \leq \kappa n^6 2^{-\tau+3}.$$

Wahrscheinlichkeit W_3 für ¬A3: Sei W_3^* die entsprechende Wahrscheinlichkeit für eines der weniger als $n^2/2$ Paare $\{P, Q\}$ und eine der weniger als $2n$ Pseudosignaturen. Da zu jeder Pseudosignatur die Akzeptionsstufe, mit der P sie akzeptieren muß, durch die Runde gegeben ist, erhält man dann

$$W_3 \leq n^3 W_3^*.$$

Bestimmung von W_3^*: In Abschnitt 3.1.6 wurde W_3^* abgeschätzt, indem die

Anzahl der Pseudosignaturen, die die Störer an P und Q in allen möglichen Protokollabläufen insgesamt ausprobieren können, abgeschätzt wurde.

Wie für Satz 3.9 begründet, sei o. B. d. A. das Störprotokoll deterministisch und P und Q seien die einzigen korrekten Prozessoren. Die Störer kennen alle $2n$ korrekten Pseudosignaturen und von allen *außer* der betrachteten die Zuordnung der Minisignaturen zu den Prozessoren. Da P und Q im Verteilungsprotokoll deterministisch arbeiten, sind die Wahrscheinlichkeiten nur noch über die Zuordnung der Minisignaturen der betrachteten Pseudosignatur zu P und Q bestimmt.

Der Strategiebaum B der Störer sieht hier, bezogen auf das konkrete Protokoll, etwas anders aus als im in Abschnitt 3.1.6.4 betrachteten schlimmsten Fall: Die Knoten des Strategiebaumes entsprechen den Pseudosignaturen, die P und Q in einer Runde von den Störern erhalten. Jeder Weg in B kann daher höchstens $2h(n,\sigma) \le 4n$ Pseudosignaturen enthalten.

Gesucht ist daher die Anzahl der verschiedenen Wege in B. Ist diese w, so ist

$$W_3^* \le w\ 4n\ 2^{-\kappa}.$$

Der Strategiebaum hat die Tiefe $t \le n$, entsprechend den Runden [0] (≈ Wurzel) bis [t] (≈ Blätter) des Protokolls. Auf jeden Knoten der Tiefe $k < t$ können P und Q in Runde [k+1] reagieren, indem sie keine, eine oder zwei Nachrichten weiterleiten. Da alle Pseudosignaturen von P und Q den Störern schon bekannt sind, erhalten sie aus den neuen Pseudosignaturen keine Information. Ein Knoten hat daher ebensoviele Kinder, wie P und Q Möglichkeiten haben, keine, eine oder zwei Nachrichten weiterzuleiten. Das Kind, das der Reaktion „weder P noch Q leiten eine Nachricht weiter“ entspricht, sei jeweils das äußerst linke.

Laut Protokoll leiten P und Q jeder höchstens zwei Nachrichten weiter. Betrachtet man also einen Weg von der Wurzel zu einem Blatt (d. h. zur letzten Runde des Protokolls), so können auf diesem Weg höchstens 4 Verzweigungen (von $\le n$ insgesamt) nach rechts (d. h. nicht nach äußerst links) vorkommen. In jeder Runde akzeptiert P bzw. Q von jedem anderen Prozessor höchstens 2 Nachrichten (eine A- und eine B-signierte), insgesamt also höchstens $2n$–2.

Zu Beginn kann jeder von beiden keine, eine oder zwei Nachrichten weiterleiten. Dies sind

$$\binom{2n-2}{0} + \binom{2n-2}{1} + \binom{2n-2}{2} = 1 + 2n-2 + (n-1)(2n-3) = 2n^2 - 3n + 2 \le 2n^2$$

Möglichkeiten für P und ebensoviele für Q. Vor der ersten Abzweigung kann ein Knoten daher $\le x_0 := 4n^4$ Kinder haben. Nach jeder Verzweigung nach rechts verringern sich die Reaktionsmöglichkeiten von P und Q, denn einer von ihnen wird nun einmal weniger reagieren als zuvor:

Nach der ersten hat einer immer noch $\le 2n^2$ Möglichkeiten, der andere aber nur noch $\le 2n$, im Produkt also $\le x_1 := 4n^3$.

Nach der zweiten hat nun entweder der eine nur noch eine Möglichkeit und

der andere $2n^2$, oder beide haben nur noch $2n$, im Produkt also stets $\le x_2 := 4n^2$.

Nach der dritten hat einer nur noch eine Möglichkeit, der andere nur noch $\le x_3 := 2n$. (Nach der vierten haben beide nur noch eine Möglichkeit.)

Damit erhält man eine obere Abschätzung der Anzahl der Wege: Man betrachtet, in welchen der n–1 Tiefen die $L \le 4$ Verzweigungen nach rechts auftreten; hiervon gibt es $\le \binom{n}{L}$ Möglichkeiten. Eindeutig bestimmt wird ein Weg dann, wenn man dazu noch angibt, welche der x_0–1 Abzweigungen nach rechts beim ersten Mal eingeschlagen wird, welche der x_1–1 beim zweiten Mal etc. Also gibt es insgesamt

$$w_L \le \binom{n}{L} x_0 \dots x_{L-1},\ L = 0, 1, 2, 3, 4$$

Wege mit L Verzweigungen nach rechts. Durch einfaches Nachrechnen erhält man so für die Anzahl w der Wege

$$\begin{aligned} w &\le \binom{n}{0} + \binom{n}{1} x_0 + \binom{n}{2} x_0 x_1 + \binom{n}{3} x_0 x_1 x_2 + \binom{n}{4} x_0 x_1 x_2 x_3 \\ &\le \binom{n}{0} + \binom{n}{1} 4n^4 + \binom{n}{2} 16n^7 + \binom{n}{3} 64n^9 + \binom{n}{4} 128n^{10} \\ &\le 1 + 4n^5 + 8n^9 + \frac{32}{3} n^{12} + \frac{16}{3} n^{14} \end{aligned}$$

Für $n \ge 2$ folgt hieraus $w \le 9n^{14}$.

Also ist $W_3^* \le w\ 4n\ 2^{-\kappa} \le 36\ n^{15}\ 2^{-\kappa}$

und damit $W_3 \le n^3\ W_3^* \le 36\ n^{18}\ 2^{-\kappa}$.

Zusammensetzen zu W:

$$W \le W_1 + W_2 + W_3 \le \kappa(8n^7+8n^6)\,2^{-\tau} + 36\ n^{18}\ 2^{-\kappa}$$

Beide Summanden sind höchstens $2^{-\sigma-1}$:

1. Summand: Es ist $8n^7 + 8n^6 \le 12n^7$ für $n \ge 2$.

$\kappa(8n^7 + 8n^6)\,2^{-\tau} \le 12\ \kappa n^7\,2^{-\tau} \le 2^{-\sigma-1} \Leftarrow \tau \ge \sigma + \log_2(\kappa) + 7\log_2(n) + \log_2(24)$.

2. Summand:

$$36\ n^{18}\ 2^{-\kappa} \le 2^{-\sigma-1} \Leftrightarrow \kappa \ge \sigma + 18\log_2(n) + \log_2(72).$$ ❒

3.2.3 Polynomial viele Verteilungen mit logarithmischer Vorbereitung

Protokoll 3.6 benötigt, wie die Protokolle in Kapitel 2, ein Vorbereitungsprotokoll mit sicheren, nichtabhörbaren Verbindungen und einem zuverlässigen Verteilkanal.

Die Länge z der Nachrichten, die verteilt werden sollen, geht in den Aufwand des Vorbereitungsprotokolls nur logarithmisch ein (Lemma 3.4d). Mithin kann mit einem Vorbereitungsprotokoll der Transmitter eine in n und σ polynomial lange Nachricht zuverlässig verteilen.

Für u beabsichtigte Ausführungen des Verteilungsprotokolls 3.6 muß das zugehörige Vorbereitungsprotokoll u mal ausgeführt werden.

Würde man das Verteilungsprotokolls mit üblichen, nur kryptographisch sicheren digitalen Signaturen implementieren, so könnte man nach einer „kurzen“ Schlüsselaustauschphase eine im kryptographischen Sicherheitsparameter polynomiale Anzahl u an Verteilungen durchführen, vereinfacht gesagt also fast beliebig viele.

Dies ist im Prinzip auch hier möglich: Es genügt, wenn zu Beginn, d.h. wenn ein physikalisch zuverlässiger Verteilkanal zur Verfügung steht, eine von u unabhängige Anzahl an Verteilungen vorbereitet wird. Diese können später dann so verwendet werden, daß trotzdem u Verteilungen durchgeführt werden können. Die Anzahl u geht in den Aufwand des gesamten Protokolls nur über den Sicherheitsparameter σ ein; dieser ist logarithmisch in u.

Grundidee: Durch das Verteilungsprotokoll 3.6 kann ein zuverlässiger Verteilkanal für fast beliebig lange Nachrichten implementiert werden. Die Grundidee des folgenden ist daher, das Verteilungsprotokoll selbst dazu zu verwenden, das Vorbereitungsprotokoll für weitere Verteilungen auszuführen.

Man beginnt also mit einer gewissen Anzahl ϑ von vorbereiteten Ausführungen von Protokoll 3.6 (für passend lange Nachrichten, s.u. Schritt 4). Wird das Verteilungsprotokoll ausgeführt, so wird dabei eine der vorbereiteten Ausführungen „verbraucht“. Diese wird ersetzt, indem man das Vorbereitungsprotokoll parallel mehrfach ausführt. Die notwendige zuverlässige Verteilung wird durch die noch unverbrauchten vorbereiteten Ausführungen realisiert, und zwar so, daß dabei soviele neue Ausführungen parallel vorbereitet werden, daß diese Erneuerung im Prinzip beliebig oft wiederholt werden kann. Seiner Funktionsweise entsprechend wird dieses Protokoll hier **Erneuerungsprotokoll** genannt.

Begrenzt wird die Anzahl u der damit möglichen byzantinischen Verteilungen nur dadurch, daß die Fehlerwahrscheinlichkeit exponentiell klein bleiben muß. Hierzu muß u logarithmisch in den Sicherheitsparameter σ von Protokoll 3.6 eingehen (s.u. Schritt 5).

Diese Grundidee wird in fünf Schritten genauer beschrieben:

Zunächst wird beschrieben, wieviele und wie lange Nachrichten verteilt werden müssen, um w Pseudosignaturen für beliebiges w parallel zu initialisieren.

1. Parallele Initialisierung von w Pseudosignaturen: Am Ende von Abschnitt 3.1.3.3.2, im Anschluß an das Initialisierungsprotokoll, wurde bereits bemerkt, daß man ohne wesentliche Erhöhung der Nachrichtenanzahl beliebig viele, etwa w, Pseudosignaturen parallel initialisieren kann. Man muß hierzu in Protokoll 3.4 nur wm statt m Grundinitialisierungen parallel vornehmen. Es stört dabei auch nicht, wenn verschiedene Prozessoren die Rolle des Signierers spielen.

Von Interesse sind hier nur die Nachrichten, die zuverlässig verteilt werden müssen, d.h. für die später Protokoll 3.6 verwendet werden soll. Man rechnet leicht nach:

Phase 1: Es werden keine Nachrichten verteilt.

Phase 2: Jeder Prozessor verteilt höchstens zwei Nachrichten, eine, um seine Authentikationsschlüssel zu senden, und ggf. eine, um sich zu beschweren. Dies sind

$$2n \text{ Nachrichten, jede der Länge} \leq 2\ wmn(\alpha(z,\tau)+1) \text{ bit.}$$

Phase 3: Jeder Prozessor verteilt genau eine Nachricht. Diese bezieht sich im schlimmsten Fall auf alle δm parallelen Ausführungen in Phase 2. Man sieht leicht:

$$n \text{ Nachrichten, jede der Länge } 2\ wmn^2(\alpha(z,\tau)+1) \text{ bit.}$$

Die n Nachrichten in Phase 3 entstehen nur im Falle einer Störung. Da Phase 2 wiederholt werden muß, kostet jede Störung $3n$ Nachrichten, alle $< n^2/2$ Störungen zusammen $< 3n^3/2$ Nachrichten. ♦

Die w vorbereiteten Pseudosignaturen sollen letztendlich dazu dienen, das Verteilungsprotokoll 3.6 w^* mal ausführen zu können, für passendes w^*. Als nächster Schritt wird daher bestimmt, wie sich w aus w^* ergibt.

2. Parallele Vorbereitung von w^* Verteilungen: Für jede Ausführung von Protokoll 3.6, Phase 2, müssen $2n-1$ Pseudosignaturen initialisiert werden. Für w^* Verteilungen genügt es daher, wenn $w = 2nw^*$ Pseudosignaturen vorbereitet werden. Aus dem oben Gesagten folgt damit:

Wird die Vorbereitung der Pseudosignaturen nicht gestört, so müssen nach (1.)

$$2n \text{ Nachrichten, jede der Länge} \leq 4\ w^* mn^2(\alpha(z,\tau)+1) \text{ bit}$$

zuverlässig verteilt werden. Für die höchstens $n^2/2$ Störungen müssen insgesamt

$$n^3 \text{ Nachrichten, jede der Länge} \leq 4\ w^* mn^2(\alpha(z,\tau)+1) \text{ bit, und}$$

$$< n^3/2 \text{ Nachrichten, jede der Länge } 4\ w^* mn^3(\alpha(z,\tau)+1) \text{ bit}$$

zuverlässig verteilt werden. ♦

Als dritter Schritt werden nun die Anzahlen ϑ und w^* bestimmt.

3. Bestimmung von ϑ und w^*: Zur Vorbereitung werden unter Verwendung eines physikalisch zuverlässigen Verteilkanals ϑ Ausführungen des Verteilungsprotokolls 3.6 vorbereitet. ϑ muß so groß sein, daß wenn das Verteilungsprotokoll ausgeführt werden soll,

- die gewünschte Verteilung durchgeführt werden kann, und
- zugleich parallel hinreichend viele neue Verteilungen vorbereitet werden können.

Nach dem oben Gesagten genügt hierfür

$$\vartheta := 1 + 2n + \lfloor 3n^3/2 \rfloor.$$

Hiervon dienen $\lfloor 3n^3/2 \rfloor$ Vorbereitungen der Störungsbehandlung und sind eine einmalige Investition, während $2n+1$ jeweils nach Verbrauch wieder ersetzt werden müssen. Damit ist also

$$w^* = 2n+1. \qquad \blacklozenge$$

Als vierter Schritt wird die Maximallänge z der zu verteilenden Nachrichten bestimmt. Sei hierzu z^+ die vorgegebene Länge der zu verteilenden echten Nachrichten.

4. Maximallänge z der zu verteilenden Nachrichten: Der Einfachheit halber wird angenommen, daß für alle vorzubereitenden Verteilungen dieselbe Maximallänge z gewählt wird. Die längsten der oben vorkommenden Nachrichten sind die zur Störungsbehandlung. Es ist daher notwendig

$$z \geq \max\{z^+, 4w^* m n^3(\alpha(z,\tau)+1)\}.$$

Nach dem oben Gesagten und nach Satz 3.6d ist

$$w^* = 2n+1 \leq 3n \quad \text{und} \quad m \leq 2\lambda^2\kappa \leq 2n^2\kappa,$$

nach Abschnitt 3.1.2 ist für $\tau \geq 1$

$$\alpha(z,\tau)+1 = 4\left\lceil \log_2\left(\frac{z}{\tau+1}\right)\right\rceil (\tau + 1 + \lceil \log_2(\log_2(z))\rceil) + 1$$

$$\leq 4 \log_2(z)\ (\tau + 3 + \log_2(\log_2(z))).$$

Für $x > 0$ ist $\log_2(x) \leq \sqrt{x} + 1$ (wie man leicht nachrechnet), und damit

$$\alpha(z,\tau)+1 \leq 4 \log_2(z)\ (\tau + 4 + \sqrt{\log_2(z)}).$$

Insgesamt ist daher für $z \geq 4w^* m n^3(\alpha(z,\tau)+1)$ hinreichend

$$z \geq 3 \cdot 2^5\, n^6\, \kappa \log_2(z)\ (\tau + 4 + \sqrt{\log_2(z)})$$

$$= \left(3 \cdot 2^5\, n^6\, \kappa \log_2(z)\ (\tau + 4)\right) + \left(3 \cdot 2^5\, n^6\, \kappa \log_2(z)^{3/2}\right)$$

Es ist hinreichend, wenn jeder der beiden Summanden $\leq z/2$ ist. Für $z \geq 1000$ gilt (wie man leicht nachrechnet)

$$\log_2(z) \leq \sqrt[3]{z},$$

1. Summand:

$$z/2 \geq 3 \cdot 2^5\, n^6\, \kappa \log_2(z)\ (\tau + 4) \Leftarrow z \geq 3 \cdot 2^6\, n^6\, \kappa\, (\tau+4)\, \sqrt[3]{z}$$

$$\Leftrightarrow z \geq 3^{3/2}\, 2^9\, n^9\, \kappa^{3/2}\, (\tau+4)^{3/2}$$

2. Summand:

$$z/2 \geq 3 \cdot 2^5\, n^6\, \kappa \log_2(z)^{3/2} \Leftarrow z/2 \geq 3 \cdot 2^5\, n^6\, \kappa \sqrt{z}$$

$$\Leftrightarrow z \geq 3^2\, 2^{12}\, n^{12}\, \kappa^2.$$

Damit ist hinreichend

$$z \geq 3^2\, 2^{12}\, n^{12}\, \kappa^2\, (\tau{+}4)^{3/2}.$$

Man wählt daher

$$z := \max\{z^+, \lceil 3^2\, 2^{12}\, n^{12}\, \kappa^2\, (\tau{+}4)^{3/2} \rceil\}.$$

Die großen Potenzen in z sind größtenteils eine Folge der recht groben Abschätzungen. Sie stören hier nicht, da z in den Aufwand des Vorbereitungsprotokolls nur logarithmisch eingeht.

Verwendet man eine möglichst kurze Codierung von Nachrichten, so geht in den Aufwand des Verteilungsprotokolls nur die *reale* Länge der Nachrichten linear ein, und diese dürfte im allgemeinen sehr viel kleiner als z sein. ♦

Als fünfter Schritt wird nun die Fehlerwahrscheinlichkeit des Erneuerungsprotokolls bestimmt, wenn es u mal ausgeführt wird.

5. Fehlerwahrscheinlichkeit für u Ausführungen: Die Fehlerwahrscheinlichkeit ergibt sich aus der Anzahl a der Ausführungen von Protokoll 3.6 und dessen Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ als $\leq a\, 2^{-\sigma}$.

Zur Störungsbeseitigung werden weniger als $3n^3/2$ Nachrichten zuverlässig verteilt. Ansonsten kostet jede der u Ausführungen $2n{+}1$ Verteilungen, zusammen also (mit $n \geq 2$, und damit $2n{+}1 \leq n^3$)

$$a \leq 3n^3/2 + (2n{+}1)u \leq 3n^3/2 + 3nu.$$

Der Fehler ist $\leq 2^{-\delta}$ für einen Sicherheitsparameter δ, wenn gilt

$$3n^3\, 2^{-\sigma-1} + (2n{+}1)u\, 2^{-\sigma} \leq 2^{-\delta}.$$

Beide Summanden werden so abgeschätzt, daß sie $\leq 2^{-\delta-1}$ sind:

$$3n^3\, 2^{-\sigma-1} \leq 2^{-\delta-1} \Leftrightarrow \sigma \geq \delta + 3\log_2(n) + \log_2(3)$$

$$3nu\, 2^{-\sigma} \leq 2^{-\delta-1} \Leftrightarrow \sigma \geq \delta + \log_2(n) + \log_2(u) + \log_2(3),$$

d. h. es ist hinreichend

$$\sigma := \lceil \delta + \log_2(3) + \log_2(n) + \max\{2\log_2(n), \log_2(u)\} \rceil.$$

Die Anzahl u geht nur über den Parameter σ in den Aufwand von Protokoll 3.6 ein, d. h. nur logarithmisch.[34] ♦

Insgesamt erhält man folgenden Satz:

Satz 3.12: (Erneuerungsprotokoll)

Sei t die Anzahl der zu tolerierenden Störer, u die Anzahl der beabsichtigten Verteilungen, z^+ die Länge der zu verteilenden Nachrichten und δ ein

---

[34] Im 4. Schritt wurde $w^* = 2n{+}1 \leq 3n$ abgeschätzt. Man kann daher mit dem obigen Wert für z stets $3n$ Verteilungen parallel vorbereiten. Tut man dies, so können $n{-}1$ Verteilungen (statt nur einer) durchgeführt werden, bevor wieder $3n$ neue vorbereitet werden müssen. In diesem Fall erhält man $a \leq 3n^2/2 + u + 3nu/(n{-}1) \leq 4n^2 + 8u$. Es genügt dann

$$\sigma \geq \lceil \delta + 3 + \max\{\log_2(u) + 1, 2\log_2(n)\} \rceil.$$

Sicherheitsparameter.

Sei hierzu

$$\sigma := \lceil \delta + \log_2(3) + \log_2(n) + \max\{2\log_2(n), \log_2(u)\} \rceil,$$

$$z := \max\{z^+, \lceil 3^2\, 2^{12}\, n^{12}\, \kappa^2\, (\tau+4)^{3/2} \rceil\},$$

wobei κ und τ wie in Protokoll 3.6 bestimmt sind. Dann gilt:

a) Werden für Protokoll 3.6 mit Parameter t, z und σ

$$\vartheta := 1 + 2n + \lfloor 3n^3/2 \rfloor$$

Ausführungen vorbereitet, und stehen auch für die Ausführung von Protokoll 3.6, Phase 2 sichere, nichtabhörbare Verbindungen zur Verfügung, so können damit in der oben beschriebenen Art u byzantinische Verteilungen für Nachrichten der Länge $\le z$ durchgeführt werden. Der Aufwand ist insgesamt polynomial in n, u, z und δ.

b) Für jede byzantinische Verteilung muß Protokoll 3.6, Phase 2 wenigstens $2n$ mal ausgeführt werden und werden wenigstens $2(t+1)$ Runden benötigt. Der zu verteilende Wert steht deterministisch nach $t+1$ Runden fest.

Für alle u Verteilungen werden im schlimmsten Fall $\le 2nu+3n^3/2$ Ausführungen von Protokoll 3.6 und hierfür $\le (2u + 3n^2/2)(t+1)$ Runden benötigt.

c) Die Fehlerwahrscheinlichkeit für alle u Verteilungen ist $\le 2^{-\delta}$. ♦

Beweis

a) Wurde vor Satz 3.12 bewiesen.

b) Protokoll 3.6, Phase 2, benötigt deterministisch $t+1$ Runden. Damit steht der zu verteilende Wert nach $t+1$ Runden fest.

Parallel hierzu werden $2n+1$ neue Verteilungen vorbereitet. Wie im 2. Schritt erläutert, muß hierzu Protokoll 3.6 $2n$ mal ausgeführt werden, wobei jeweils n Verteilungen parallel erfolgen. Dies kostet daher $2(t+1)$ Runden.

Zur Störungsbeseitigung müssen im schlimmsten Fall $3n^2/2$ mal jeweils n Nachrichten verteilt werden. Dies kostet daher $(3n^2/2)(t+1)$ Runden, insgesamt also $(2u + 3n^2/2)(t+1)$ Runden.

c) Obiger 5. Schritt. ❒

Im hier skizzierten Erneuerungsprotokoll geht die Anzahl u der beabsichtigten Ausführungen nur logarithmisch in den Aufwand des Vorbereitungs- und Verteilungsprotokolls ein. Bei einer exponentiell kleinen Fehlerwahrscheinlichkeit kann man prinzipiell kein besseres Resultat erwarten.

3.3 Zusammenfassung

In diesem Kapitel wurde das erste byzantinische Verteilungsprotokoll

beschrieben, das im byzantinischen Fehlermodell jedes $t < n$ toleriert.

Grundlage waren *Pseudosignaturen*, d.h. ein neuartiger Authentikationsmechanismus, der in vielen Protokollen statt herkömmlicher digitaler Signaturen verwendet werden kann, aber im Gegensatz zu diesen auch im byzantinischen Fehlfunktionsmodell beweisbar sicher ist (Abschnitt 3.1). Pseudosignaturen sind eine Verallgemeinerung eines von CHAUM und ROIJAKKERS in [ChRo_91] vorgeschlagenen Verfahrens.

Mit einer Pseudosignatur kann jeweils ein Wert so signiert werden, daß er maximal λ mal weitergegeben werden kann, für ein frei wählbares λ.

Die Fehlerwahrscheinlichkeit von Pseudosignaturen hängt vom Protokoll ab, in dem sie eingesetzt werden. Für geeignete Protokolle ist sie exponentiell klein. Geeignet sind insbesondere alle hier sogenannten ordentlichen Protokolle. Dies sind im wesentlichen polynomiale Protokolle, die auf Pseudosignaturen nur über die Grundverfahren zum Initialisieren, Signieren und Testen zugreifen, und die mit Pseudosignaturen jeder Größe implementiert werden können (Abschnitt 3.1.6.1). Pseudosignaturen können daher in vielen polynomialen authentisierten Protokollen zur byzantinischen Übereinstimmung (und zur Lösung ähnlicher Probleme) verwendet werden.

Pseudosignaturen benötigen ein Vorbereitungsprotokoll, in dem ein zuverlässiger Verteilkanal und sichere, nichtabhörbare Verbindungen zur Verfügung stehen.

Das eigentliche Verteilungsprotokoll basiert auf einem authentisierten Verteilungsprotokoll von DOLEV und STRONG, das so modifiziert wurde, daß es mit Pseudosignaturen sicher implementiert werden kann.

Wie das Original benötigt das Verteilungsprotokoll deterministisch $t+1$ Runden, in denen hier für eine Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ in $O(n^2)$ Nachrichten $O(n^2 z + n^6(\sigma+\log(n))^2)$ bit über sichere, nicht notwendigerweise nichtabhörbare, Verbindungen gesendet werden müssen. z ist die Länge der zu verteilenden Nachricht.

In einem Vorbereitungsprotokoll müssen jeweils $2n-1$ Pseudosignaturen initialisiert werden.

Indem man das Verteilungsprotokoll selbst dazu verwendet, die für das nächste Vorbereitungsprotokoll notwendige zuverlässige Verteilung von Nachrichten zu realisieren, erhält man ein Verteilungsprotokoll, das für u Ausführungen nur ein in u logarithmisches Vorbereitungsprotokoll benötigt (Abschnitt 3.2.3).

4 Erweiterung binärer Verteilung auf lange Nachrichten

In Kapitel 2 wurde ein Protokoll beschrieben, das für $t < n/2$ das byzantinische Verteilungsproblem in einer erwartet konstanten Rundenanzahl löst (Protokoll 2.9). Mit diesem Protokoll können jedoch nur binäre Nachrichten verteilt werden. Soll eine z bit lange Nachricht verteilt werden, so muß man das Protokoll z mal ausführen.

Zum einen erhöht sich dadurch die Fehlerwahrscheinlichkeit von $\leq 2^{-\sigma}$ auf $\leq z\, 2^{-\sigma}$. Soll die Fehlerwahrscheinlichkeit $\leq 2^{-\omega}$ sein, so muß daher der Parameter σ passend erhöht werden:

$$z\, 2^{-\sigma} \leq 2^{-\omega} \Leftrightarrow \sigma \geq \omega + \log_2(z).$$

Zum anderen wächst der Nachrichtenaufwand auch des Vorbereitungsprotokolls linear mit z. Letzteres kann mit Hilfe der Pseudosignaturen aus Abschnitt 3.1 und eines authentisierten Protokolls von TOUEG [Toue_84] verbessert werden. In den Nachrichtenaufwand des Vorbereitungsprotokolls geht z dann nur noch logarithmisch ein.

In Abschnitt 4.1 wird das Teilprotokoll von TOUEG, das Signaturen verwendet, mit Pseudosignaturen implementiert und seine Sicherheit bewiesen. In Abschnitt 4.2 wird es in TOUEGs Protokoll eingesetzt und mit Protokoll 2.9 kombiniert; das Ergebnis terminiert in erwartet höchstens 23 Runden.

In Abschnitt 4.3 wird gezeigt, wie (analog zu Abschnitt 3.2.3) das Protokoll aus Abschnitt 4.2 selbst dazu verwendet werden kann, das Vorbereitungsprotokoll für die nächsten Verteilungen auszuführen.

4.1 Byzantinische Pseudoverteilung

TOUEGs Protokoll basiert auf einer Abschwächung der byzantinischen Verteilung, hier **byzantinische Pseudoverteilung** (*crusader agreement*) genannt. Diese ist definiert durch [Dole_82]:

Terminierung	Nach einer a priori bekannten endlichen Zeit terminiert das Protokoll.
Korrektheit	War der Transmitter korrekt und hatte den Anfangswert v, so akzeptiert jeder korrekte Prozessor den Wert v als Endwert.
Pseudokonsistenz	Alle korrekten Prozessoren, die einen Endwert akzeptieren, akzeptieren denselben Endwert.

Im Gegensatz zur byzantinischen Verteilung garantiert die byzantinische Pseudoverteilung nicht, daß alle korrekten Prozessoren einen Endwert erhalten. Es ist aber sicher, daß alle Prozessoren, die einen Endwert erhalten, denselben erhalten.

Das folgende Protokoll 4.1 realisiert Pseudoverteilung für beliebig lange Nachrichten v für $t < n/2$. Zu dem entsprechenden Protokoll aus [Toue_84] wurden die Akzeptionsstufen hinzugefügt und, da überflüssig, alle Signaturen außer der des Transmitters weggelassen.

Protokoll 4.1: Byzantinische Pseudoverteilung mit Pseudosignaturen mit Parametern z, ω

Parameter z gibt die Länge des zu verteilenden Wertes an. Zum Sicherheitsparameter ω seien

$$\kappa := \lceil \omega + 2 \log_2(n) + 1 \rceil$$
$$\tau := \lceil \omega + \log_2(\kappa) + 4 \log_2(n) + 4 + \log_2(3) \rceil.$$

Phase 1: Vorbereitungsprotokoll

Mit Phase 1 von Schema 3.5 wird eine Pseudosignatur mit Parametern $\lambda = 2$, z, τ und κ initialisiert, wobei der Transmitter T als Signierer agiert.

Phase 2: Pseudoverteilung

Der Transmitter T hat als lokale Eingabe seinen Anfangswert $v \in \{0,1\}^z$.

[1] T bestimmt mit Phase 2 von Schema 3.5 die Pseudosignatur ψ unter v und sendet an alle das Paar

$$(v, \psi).$$

[2] Jeder Prozessor P_i, der ein Paar $(v^{(i)}, \psi^{(i)})$ erhalten hat, das er 1-akzeptiert, sendet an alle die Nachricht

$$(\text{„Echo“}, i, (v^{(i)}, \psi^{(i)})).$$

[3] Jeder Prozessor P_i bildet die Menge S_i aller Tripel („Echo“, j, $(v^{(j)}, \psi^{(j)})$), die er erhalten hat und für die er $(v^{(j)}, \psi^{(j)})$ 2-akzeptiert.

Enthalten alle Nachrichten in S_i denselben Wert v, und ist $|S_i| > n/2$, so erhält P_i als Endwert $v_i := v$. Ansonsten erhält er $v_i := \varepsilon$ (das leere Wort).

In Satz 4.2 wird gezeigt, daß Protokoll 4.1 mit Fehlerwahrscheinlichkeit höchstens $2^{-\omega}$ das byzantinische Pseudoverteilungsproblem löst. Wie in Abschnitt 3.2 wird zuerst eine deterministische Version dieses Satzes gezeigt.

Lemma 4.1: (Deterministische Version der Pseudoverteilung)
Angenommen, für alle in einem Protokollablauf vorkommenden Pseudo-

signaturen ψ und korrekten Prozessoren P_i, P_j gilt:

A1. Ist T korrekt und ist ψ die Pseudosignatur von T unter den Wert v, so 1-akzeptiert P_i dies ψ.

A2. Ist T korrekt und 1-akzeptiert (bzw. 2-akzeptiert) P_i in Schritt [2] (bzw. in [3]) den Wert ψ als Pseudosignatur von T unter den Wert v, so hat T in Schritt [1] diese Pseudosignatur verteilt.

A3. Wird ein Wert ψ von P_i in Schritt [2] 1-akzeptiert, so wird ihn P_j in Schritt [3] 2-akzeptieren.

Ist $t < n/2$, so gilt dann für Protokoll 4.1:

a) Für alle korrekten Prozessoren P_i und P_j mit $v_i \neq \varepsilon$ und $v_j \neq \varepsilon$ gilt $v_i = v_j$.

b) Ist der Transmitter korrekt und verteilt den Wert v, so gilt für alle korrekten Prozessoren P_i am Ende $v_i = v$.

c) Der Aufwand des Protokolls ist polynomial in n, z und ω.

Im Verteilungsprotokoll werden in 2 Runden in $O(n^2)$ Nachrichten $O(n^2z + n^3(\omega+\log(n))^2)$ bit über sichere, nicht notwendigerweise nichtabhörbare Verbindungen gesendet. ♦

Beweis:

a) Angenommen, P_i erhält am Ende $v_i \neq \varepsilon$. Dann hat P_i in S_i wenigstens $n/2 > t$ Nachrichten („Echo“, k, $(v_i, \psi^{(k)})$) 2-akzeptiert. Von diesen stammte wenigstens eine von einem korrekten Prozessor P_k, der folglich $(v_i, \psi^{(k)})$ in Schritt [1] 1-akzeptierte und („Echo“, k, $(v_i, \psi^{(k)})$) an *alle* Prozessoren, auch P_j, verteilt hat. Nach (A3) 2-akzeptiert auch P_j diese Nachricht und kann daher keinen anderen Wert als v_i erhalten.

b) Ist der Transmitter korrekt, so 1-akzeptieren nach (A1) alle korrekten Prozessoren seinen Wert in Schritt [1]. Alle korrekten Prozessoren 2-akzeptieren in Schritt [2] den Wert von wenigstens $n-t > n/2$ Prozessoren, d.h. $|S_i| > n/2$. Nach (A2) akzeptiert kein korrekter Prozessor einen anderen Wert als den des Transmitters, so daß alle v erhalten.

c) Die Polynomialität ist offensichtlich, ebenso die Runden- und Nachrichtenanzahl im Verteilungsprotokoll.

Die $O(n^2)$ Nachrichten bestehen im wesentlichen jeweils aus einem Paar (v, ψ). v hat die Länge z, ψ nach Satz 3.6d die Länge $O(n\kappa\tau)$ bit = $O(n(\omega+\log(n))^2)$ bit. ❒

Satz 4.2: (Pseudoverteilung mit Pseudosignaturen)

Ist $t < n/2$, so löst Protokoll 4.1 in jedem Protokollablauf mit Wahrscheinlichkeit $\geq 1-2^{-\omega}$ das byzantinische Pseudoverteilungsproblem. ♦

Beweis: Der Beweis erfolgt analog zum Beweis von Satz 3.11, d.h. es werden die

Wahrscheinlichkeiten W_i für das Nichtzutreffen von Annahme (Ai) in Lemma 4.1 abgeschätzt und die Fehlerwahrscheinlichkeit W dann durch

$$W \le W_1 + W_2 + W_3.$$

W_1 und W_2 werden wie in Satz 3.7 bzw. Satz 3.8 abgeschätzt. Es ist $\lambda = 2$, $g(n,\omega) = 1$ und $h(n,\omega) = n+1$. Folglich ist

$$W_1 \le g(n,\omega)\lambda^2\kappa n^4 2^{-\tau+2} = \kappa\, n^4 2^{-\tau+4}$$

und $$W_2 \le g(n,\omega)\, h(n,\omega)\, \lambda^2\kappa n^2 2^{-\tau+1} = \kappa\,(n+1) n^2 2^{-\tau+3}.$$

Auf analoge Art liefert Satz 3.9 $W_3 \le n^2 3^{2n+3} 2^{-\kappa}$. Dies kann allerdings noch erheblich verbessert werden:

Für $\lambda = 2$ kann eine Nachricht höchstens falsch 1-akzeptiert werden (denn bei λ Akzeptionsstufen ist falsches λ-akzeptieren nicht möglich), d. h. höchstens die Nachricht, die der Transmitter in der ersten Runde verteilt. Folglich interessieren hier im Strategiebaum die Blätter nicht, und man kann $h(n,\omega) = 1$ verwenden und erhält $W_3 \le n^2 3^3 2^{-\kappa}$.

Auch der Faktor $3^3 = 27$ kann noch eingespart werden, denn aktive Angriffe sind hier ausgeschlossen: Alle Prozessoren erwarten vom Transmitter genau eine Nachricht, und zwar alle in der ersten Runde. Nach der Reaktion der korrekten Prozessoren in der zweiten Runde haben die Störer daher keinen weiteren Versuch. Daher genügt es, Satz 3.6c auf alle $\le n^2$ Paare von Prozessoren und die Akzeptionsstufe $k = 1$ anzuwenden:

$$W_3 \le n^2\, 2^{-\kappa}.$$

Insgesamt erhält man:

$$W \le \kappa\,(2n^4+(n+1)n^2)\, 2^{-\tau+3} + n^2\, 2^{-\kappa} \le 3\ \kappa\, n^4\, 2^{-\tau+3} + n^2\, 2^{-\kappa}.$$

Beide Summanden sind höchstens $2^{-\omega-1}$, und damit ist $W \le 2^{-\omega}$:

1. Summand:

$$3\ \kappa n^4\, 2^{-\tau+3} \le 2^{-\omega-1} \Leftrightarrow \tau \ge \omega + \log_2(\kappa) + 4\log_2(n) + 4 + \log_2(3)$$

2. Summand:

$$n^2\, 2^{-\kappa} \le 2^{-\omega-1} \Leftrightarrow \kappa \ge \omega + 2\log_2(n) + 1.$$ ❐

4.2 TOUEGs Protokoll

Mit Protokoll 4.1 und einem binären byzantinischen Verteilungsprotokoll kann nun ein byzantinisches Verteilungsprotokoll für Nachrichten $v \in \{0,1\}^z$ konstruiert werden.

Die folgenden Protokolle und Beweise sind im wesentlichen aus [Toue_84] übernommen. Ergänzt wurden die Fehlerwahrscheinlichkeiten. (In [Toue_84] wurde, wie für authentisierte Protokolle üblich, der Authentikationsmechanismus als fehlerfrei vorausgesetzt. Alle Beweise in [Toue_84] stehen daher implizit unter derselben Voraussetzung wie hier Lemma 4.1.)

Das Verteilungsprotokoll 4.3 basiert auf dem folgenden Filterprotokoll 4.2.

Für dieses wird einige Notation benötigt: Sei $\varpi := (w_1, \ldots, w_n) \in (\{0,1\}^*)^n$ und $w \in \{0,1\}^*$. Sei dann $Anzahl(w, \varpi)$ die Anzahl der Indizes i, für die gilt $w_i = w$, und sei

$$Abs_Mehrheit(\varpi) := \begin{cases} w & \text{falls } Anzahl(w, \varpi) > n/2 \\ \varepsilon & \text{sonst} \end{cases},$$

$$Rel_Mehrheit(\varpi) := \begin{cases} w & \text{falls } Anzahl(w, \varpi) > Anzahl(w_i, \varpi) \text{ für alle } w_i \neq w \\ \varepsilon & \text{sonst} \end{cases}.$$

Protokoll 4.2: TOUEGs Filterprotokoll mit Parametern z, ω

z gibt die Länge der zu „filternden" Werte an, ω bestimmt die Fehlerwahrscheinlichkeit von Protokoll 4.1. Protokoll 4.1 wird mit Parametern z, ω verwendet.

Phase 1: Vorbereitungsprotokoll

Phase 1 von Protokoll 4.1 wird $2n$ mal ausgeführt. Dabei agiert jeder Prozessor zweimal als Signierer.

Phase 2: Filterphase

Jeder Prozessor P_i hat seinen Startwert $v_i^{(0)} \in \{0,1\}^z$.

Für $k = 1, 2$:

[k] Jeder Prozessor P_i pseudoverteilt seinen Wert $v_i^{(k-1)}$ mit Protokoll 4.1, Phase 2 (d. h. mit P_i als Transmitter). Den Wert, den er von P_j erhält, nennt er $v_{j,i}^{(k)}$.

Aus allen Werten $\varpi_i^{(k)} := (v_{1,i}^{(k)}, \ldots, v_{n,i}^{(k)})$ bestimmt P_i

$$v_i^{(k)} := Abs_Mehrheit(\varpi_i^{(k)}).$$

Als Ergebnis von Protokoll 4.2 erhält P_i das Paar

$$(v_i, m_i) := (Abs_Mehrheit(\varpi_i^{(2)}), Rel_Mehrheit(\varpi_i^{(2)})).$$

Lemma 4.3: (TOUEGs Filterprotokoll)

Ist $t < n/2$, so gilt für Protokoll 4.2 mit einer Fehlerwahrscheinlichkeit höchstens $2n\, 2^{-\omega}$:

a) Beginnen alle korrekten Prozessoren P_i das Protokoll mit demselben Wert $v_i = v$, so gilt auch nach dem Protokoll für alle $v_i = v$.

b) In Schritt [k], $k \in \{1,2\}$, gilt für alle korrekten Prozessoren P_i, P_j: Ist

$$Abs_Mehrheit(\varpi_i^{(k)}) = v \quad \text{für ein } v \neq \varepsilon,$$

so ist $Abs_Mehrheit(\varpi_j^{(k)}) \in \{\varepsilon, v\}$.

c) In Schritt [2] gilt für alle korrekten Prozessoren P_i, P_j: Ist

$$Abs_Mehrheit(\varpi_i^{(2)}) = v \quad \text{für ein } v \neq \varepsilon,$$

so ist $Rel_Mehrheit(\varpi_j^{(2)}) = v$.

d) Der Aufwand des Protokolls ist polynomial in n, z und ω.

Phase 1 benötigt im besten Fall 3, im schlimmsten Fall $2n^2$ Runden. Sei $\gamma := \big(\log(z)\ (\omega+\log(n))\ (\omega+\log(n)+\log(\log(z)))\big)$. Dann müssen im schlimmsten Fall in $O(n^4)$ Nachrichten $O(n^6\gamma)$ bit über sichere, nichtabhörbare Verbindungen und in $O(n^3)$ Nachrichten $O(n^6\gamma)$ bit über einen zuverlässigen Verteilkanal gesendet werden.

In Phase 2 werden in 4 Runden $O(n^3 z + n^4(\omega+\log(n))^2)$ bit in $O(n^2)$ Nachrichten über sichere, nicht notwendigerweise nichtabhörbare Verbindungen gesendet. ♦

Beweis: Im Protokoll 4.2 wird Protokoll 4.1 $2n$ mal aufgerufen. Jeder Aufruf hat eine Fehlerwahrscheinlichkeit von höchstens $2^{-\omega}$, so daß die Gesamtfehlerwahrscheinlichkeit höchstens $2n\ 2^{-\omega}$ ist.

Angenommen, alle Unterprotokolle arbeiteten fehlerfrei:

a) Da $n-t > n/2$ korrekte Prozessoren denselben Wert pseudoverteilen, erhält nach Satz 4.2 jeder korrekte Prozessor diesen Wert als *Abs_Mehrheit* in Schritt [1], und analog auch in Schritt [2].

b) Gilt in Schritt [k], $k \in \{1,2\}$, $Abs_Mehrheit(\varpi_i^{(k)}) = v$ für ein $v \neq \varepsilon$, so heißt dies

$$Anzahl(v, \varpi_i^{(k)}) =: u > n/2.$$

P_i hat also v von u verschiedenen Prozessoren erhalten, und nach Satz 4.2 erhalten von diesen u Prozessoren alle korrekten Prozessoren entweder dasselbe v oder ε. Nach Schritt [k] gilt daher für alle korrekten Prozessoren P_j und alle Werte $v' \notin \{\varepsilon, v\}$

$$Anzahl(v', \varpi_j^{(k)}) \leq n - u < n/2.$$

Damit kann in Schritt [k] kein P_j einen Wert $v_j^{(k)} \notin \{\varepsilon, v\}$ erhalten.

c) Angenommen, P_i erhält in Schritt [2] den Wert $v \neq \varepsilon$, indem er ihn von k_1 korrekten Prozessoren und k_0 Störern erhält mit $k_1 + k_0 > n/2$.

Wegen $k_0 \leq t < n/2$ ist $k_1 \geq 1$, d.h. es gibt einen korrekten Prozessor P_j mit $Abs_Mehrheit(\varpi_j^{(1)}) = v$. Nach (b) haben daher in Schritt [1] alle korrekten Prozessoren P_j entweder dasselbe v oder ε als $v_j^{(1)}$ erhalten. In Schritt [2] wird folglich kein korrekter Prozessor ein $v' \notin \{\varepsilon, v\}$ verteilen, und wegen Lemma 4.1b erhält jeder Prozessor wenigstens k_1 mal den Wert v. Von den höchstens t Störern haben sich bereits k_0 auf v festgelegt, d.h. von diesen erhält kein korrekter Prozessor ein $v' \notin \{\varepsilon, v\}$ (Satz 4.2). Damit ist für alle korrekten P_j und alle $v' \notin \{\varepsilon, v\}$

$$Anzahl(v', \varpi_j^{(2)}) \le t - k_0 \quad \text{und} \quad k_1 \le Anzahl(v, \varpi_j^{(2)}).$$

Wegen $k_1 + k_0 > n/2 > t$ ist $t - k_0 < k_1$, d. h. es ist

$$Anzahl(v', \varpi_j^{(2)}) < Anzahl(v, \varpi_j^{(2)}),$$

und damit $$Rel_Mehrheit(\varpi_j^{(2)}) = v.$$

d) Die Polynomialität folgt aus Lemma 4.1d.

Zur Phase 1: Den Aufwand sieht man völlig analog wie in Satz 3.10. Der einzige Unterschied ist, daß dort $\lambda = O(n)$ war, während hier $\lambda = O(1)$ ist. Damit ist hier $m = O(\omega + \log(n))$ und man sieht, daß dies die Exponenten von n in den Bitanzahlen jeweils um 2 verringert.

Zur Phase 2: Folgt aus Lemma 4.1c, wobei alle jeweils n Pseudoverteilungen in Schritt [k], $k = 1, 2$, parallel ausgeführt werden. ❒

Das folgende Protokoll realisiert die byzantinische Verteilung einer Nachricht $v \in \{0,1\}^z$ mit Hilfe eines Aufrufs von Filterprotokoll 4.2 und eines binären byzantinischen Verteilungsprotokolls Π_{01} aus Kapitel 2.

Die Fehlerwahrscheinlichkeit des Verteilungsprotokolls Π_{01} sei $\le 2^{-\rho}$, und ρ sei der einzige Parameter von Π_{01}. Der Parameter L sei jeweils optimal gewählt, d. h. in Protokoll 2.8 sei $L = 2$ oder $L = 3$ (Satz 2.9c), in Protokoll 2.9 sei $L = 2$ (Satz 2.10b).

Protokoll 4.3: TOUEGs Protokoll zur byzantinischen Verteilung mit Parametern z, σ

Parameter z gibt die Länge des zu verteilenden Wertes an, σ bestimmt die Fehlerwahrscheinlichkeit. Sei

$$\omega := \lceil \sigma + \log_2(n) + 2 \rceil.$$

Phase 1: Vorbereitungsprotokoll

Π_{01} mit Parameter $\rho = \sigma + 1$ und Protokoll 4.2 mit Parametern z und ω werden vorbereitet.

Phase 2: Verteilungsprotokoll

[0] Der Transmitter T verteilt seinen Anfangswert $v \in \{0,1\}^z$. Prozessor P_i nennt den erhaltenen Wert $v_i^{(0)}$.

[1] Das Filterprotokoll 4.2 wird mit den obigen Startwerten ausgeführt. Danach hat jeder Prozessor P_i ein Paar $(v_i^{(2)}, m_i)$.
Ist $v_i^{(2)} = \varepsilon$, so sei $b_i := 0$, sonst $b_i := 1$.

[2] Das binäre byzantinische Verteilungsprotokoll Π_{01} wird ausgeführt, allerdings erst ab Phase 2, Schritt [1] (d. h. „ohne“ Transmitter). Als Startwert $v_{i,0}$ verwendet jeder Prozessor P_i seinen Wert b_i.

Erhält P_i als Endwert von Π_{01} den Wert 0, so sei sein Endwert in diesem Protokoll $v_i := \varepsilon$, ansonsten $v_i := m_i$.

Satz 4.4: (TOUEGs Protokoll zur byzantinischen Verteilung)

Ist $t < n/2$, so gilt:

a) Protokoll 4.3 löst in jedem Protokollablauf das byzantinische Verteilungsproblem mit Wahrscheinlichkeit $\geq 1 - 2^{-\sigma}$.

b) Das Protokoll ist polynomial in n, z und σ. Das Verteilungsprotokoll benötigt 4 Runden mehr als das in Phase 2, Schritt [2] verwendete binäre byzantinische Verteilungsprotokoll Π_{01}.

c) Wird Protokoll 2.9 mit $L = 2$ als Π_{01} verwendet, so gilt:

Das Vorbereitungsprotokoll benötigt im schlimmsten Fall $O(n^2)$ Runden, in denen mit $\gamma := \big(\log(z)\ (\sigma+\log(n))\ (\sigma+\log(n)+\log(\log(z)))\big)$

$$O((\sigma(\sigma + \log(n))^3+\gamma)n^6) \text{ bit}$$

zuverlässig verteilt und über sichere nichtabhörbare Verbindungen gesendet werden.

Das Verteilungsprotokoll benötigt im erwarteten Fall 23, im schlimmsten Fall $O(\sigma)$ Runden. Dabei werden $O(n^3 z + n^4\ \sigma\ (\sigma+\log(n))^2)$ bit in erwartet $O(n^2)$ und maximal $O(\sigma n^2)$ Nachrichten über sichere, nicht notwendigerweise nichtabhörbare Verbindungen gesendet. ♦

Beweis:

a) Die Fehlerwahrscheinlichkeit wird durch die Summe der Fehlerwahrscheinlichkeiten der beiden Unterprotokolle abgeschätzt, also

$$\leq 2^{-\sigma-1} + 2n2^{-\omega} \leq 2^{-\sigma-1} + 2n2^{-\sigma-2-\log_2(n)} \leq 2^{-\sigma}.$$

Angenommen, alle Unterprotokolle arbeiteten fehlerfrei:

Erhalten in Phase 2, Schritt [0] alle korrekten Prozessoren P_i denselben Wert $v \neq \varepsilon$, so erhalten nach Lemma 4.3a alle in Schritt [1] $(v_i^{(2)}, m_i) = (v, v)$ und $b_i = 1$. Mit Π_{01} erhalten in [2] alle den Endwert 1 und erhalten daher konsistent $v_i = v$.

Herrscht zu Beginn keine Konsistenz und erhält in Schritt [1] kein korrekter Prozessor einen Wert $v_i^{(2)} \neq \varepsilon$, so setzen alle $b_i = 0$. Mit Π_{01} erhalten in Schritt [2] alle den Endwert 0 und erhalten daher konsistent $v_i = \varepsilon$.

Herrscht zu Beginn keine Konsistenz und erhält in Schritt [1] ein korrekter P_i einen Wert $v_i^{(2)} \neq \varepsilon$, so gilt nach Lemma 4.3c $m_j = v_i^{(2)}$ für *alle* korrekten P_j. Π_{01} findet auf jeden Fall ein konsistentes Ergebnis, so daß alle konsistent $v_j = \varepsilon$ oder $v_j = v_i^{(2)}$ erhalten.

b) Folgt aus Satz 2.9, Satz 2.10 und Lemma 4.3d. Die Schritte [0] und [1] benötigen zusammen 5 Runden. In Π_{01} entfällt die erste Runde, die Verteilung der Anfangswerte durch den Transmitter, da diese hier in Schritt [1] lokal bestimmt werden. Insgesamt erhöht sich die Rundenanzahl daher um 4.[35]

c) Zum Vorbereitungsprotokoll: Dieses besteht aus dem von Protokoll 2.9 und Protokoll 4.2. Ersteres benötigt $O(n)$ Runden, in denen $O(\sigma(\sigma + \log(n))^3 n^6)$ bit zuverlässig verteilt und über sichere nichtabhörbare Verbindungen gesendet werden. Letzteres ist in Lemma 4.3d angegeben. Zusammen ergeben sich die angegebenen Werte.

Zum Verteilungsprotokoll:

Protokoll 2.9 terminiert nach Satz 2.10 für $L = 2$ in erwartet 19 Runden, und im schlimmsten Fall nach Satz 2.9 in $O(\sigma)$ (mit $K = O(\sigma)$). Nach (b) benötigt daher Protokoll 4.3, Phase 2 erwartet 23 Runden und im schlimmsten Fall $O(\sigma)$.

Zu den Nachrichtenanzahlen und -längen:

[0] $O(n)$ Nachrichten der Länge z, also $O(nz)$ bit.

[1] Nach Lemma 4.3d in $O(n^2)$ Nachrichten $O(n^3 z + n^4 (\sigma+\log(n))^2)$ bit.

[2] Nach Satz 2.10, für $\sigma \geq 5$, in erwartet $O(n^2)$ und maximal $O(\sigma n^2)$ Nachrichten $O(\sigma(\sigma+\log(n))^2 n^4)$ bit.

Hieraus folgen die im Satz angegebenen Werte. ❒

4.3 Polynomial viele Verteilungen mit logarithmischer Vorbereitung

In Abschnitt 3.2.3 wurde beschrieben, wie u Verteilungen so durchgeführt werden können, daß u in den Aufwand der Vorbereitung nur logarithmisch eingeht.

Die Grundidee war, nur wenige Verteilungen im eigentlichen Vorbereitungsprotokoll (d.h. mit zuverlässigem Verteilkanal) vorzubereiten. Wird eine Nachricht später verteilt, so werden parallel mehrere neue Verteilungen vorbereitet, um dabei verloren gegangene vorbereitete Verteilungen zu ersetzen. Die hierzu notwendige zuverlässige Verteilung wird durch die vorbereiteten Protokollausführungen realisiert.

Dies ergibt allerdings nur dann einen Sinn, wenn dabei mehr neue Protokollausführungen vorbereitet als verbraucht werden, und dies setzt voraus, daß man in einer Protokollausführung (fast) beliebig lange Nachrichten zuverlässig verteilen kann.

Dieser Ansatz ist auf *binäre* Verteilungsprotokolle wie die in Kapitel 2 daher nicht anwendbar. Auf Protokoll 4.3, implementiert mit Protokoll 2.9, kann er

---

[35] Ohne die Anfangsrunde bildet Π_{01} gerade ein Protokoll zur byzantinischen Übereinstimmung. Auf dieselbe Art und mit demselben Rundenmehraufwand kann man TOUEGs Protokoll mit jedem binären byzantinischen Übereinstimmungsprotokoll implementieren.

jedoch völlig analog angewandt werden:

Zunächst ist klar, daß man mehrere Ausführungen von Protokoll 4.3 *parallel* vorbereiten kann (vgl. 1. und 2. Schritt in Abschnitt 3.2.3).

Dies benötigt im ungestörten Fall O(1) Runden. Zur Störungsbeseitigung kommen $O(n^2)$ Runden hinzu, wobei diese Runden allerdings für beliebig viele Iterationen nur einmal anfallen (vgl. Satz 2.9 und Abschnitt 2.1.2.5 bzw. Satz 3.5 und nachfolgende Bemerkungen).

Faßt man alle von einem Prozessor in einer Runde zu verteilenden Nachrichten zu einer zusammen, so kostet jede Runde höchstens n Verteilungen. Im ungestörten Fall werden daher $\alpha = O(n)$ Verteilungen benötigt, und zur Störungsbeseitigung insgesamt $\beta = O(n^3)$.

Wird dies u mal iteriert, so werden insgesamt $u\alpha + \beta = O(un+n^3)$ Verteilungen benötigt. Von diesen finden jeweils n Verteilungen parallel statt.

Nach einer ersten Vorbereitung mittels zuverlässigem Verteilkanal kann jede dieser Verteilungen mit Protokoll 4.3 realisiert werden.

Hierzu muß man nur den Parameter z, d.h. die Maximallänge der Nachrichten, die verteilt werden können, passend groß wählen. Dies ist möglich, da z selbst in das Vorbereitungsprotokoll nur logarithmisch eingeht (über die Länge $\alpha(z,\tau)$ der Authentikationsschlüssel, die für die Pseudosignaturen verwendet werden; vgl. den 4. Schritt in Abschnitt 3.2.3).

Die synchrone Rundeneinteilung erfordert, daß jeder Prozessor stets weiß, wann er eine neue Runde beginnen kann, d.h. wann alle Nachrichten der letzten Runde spätestens bei ihm eingetroffen sein müssen. Wird Protokoll 2.9 in Protokoll 4.3 verwendet, so endet jede Verteilung zwar nach erwartet 23 Runden. Über die Terminierung herrscht aber in Protokoll 2.9 nicht notwendigerweise Einigkeit. Am einfachsten wird dieses Problem gelöst, indem man für jede Verteilung die maximalen $O(\sigma)$ Runden vorsieht.[36]

Jede Verteilrunde, bestehend aus bis zu n parallelen Ausführungen von Protokoll 4.3, dauert daher $O(\sigma)$ echte Runden.

---

[36] Man könnte jedem Prozessor P_i auch erlauben, nachdem in Runde r aus seiner Sicht eine Verteilrunde terminiert hat, in Runde r+1 die nächste zu beginnen. Hierzu sind zwei Probleme zu lösen:

Zum einen ist dies nur sinnvoll, wenn P_i auf nichttriviale Art (d.h. besser als $O(\sigma)$) abschätzen kann, wann spätestens alle anderen Prozessoren die aktuelle Verteilrunde ebenfalls beenden. Man sieht relativ einfach, daß P_i dies in Protokoll 2.9 durch genaueres Auswerten und weiteres „Mithören" der Münzwürfe recht gut abschätzen kann (vgl. Schluß von Abschnitt 2.2.3).

Zum anderen kann P_i in Runde r+1 die nächste Verteilrunden nur beginnen, wenn *alle* Prozessoren wissen, daß in Runde r+1 ein Verteilungsprotokoll beginnt (*distributed firing squad problem* [CDDS_85]). Auf recht brutale Art kann man dieses Problem lösen, indem man in *jeder* echten Runde für *jeden* Prozessor als Transmitter ein Verteilungsprotokoll beginnen läßt.

Eine Verteilung sieht damit, wie in Abschnitt 3.2.3, wie folgt aus (vgl. den 3. Schritt in Abschnitt 3.2.3):

Vorbereitung mittels physikalisch zuverlässigem Verteilkanal:
Es werden mit Protokoll 4.3, Phase 1,

$$\vartheta := \alpha+\beta+1 = O(n^3)$$

Verteilungen passender Länge z vorbereitet.

Verteilung einer Nachricht v:

- Eine der vorbereiteten Verteilungen wird dazu verwendet, v zu verteilen. (Damit ist v nach erwartet 23 Runden verteilt.)
- Parallel dazu wird das Vorbereitungsprotokoll, d. h. Protokoll 4.3, Phase 1, für $\alpha+1$ Verteilungen parallel ausgeführt.
  Der zuverlässige Verteilkanal wird durch die noch nicht verbrauchten vorbereiteten Ausführungen von Protokoll 4.3 realisiert.

Man beachte dabei, daß die β Verteilungen zur Störungsbeseitigung nur einmal anfallen und daher nicht ersetzt werden müssen.

Dieses Protokoll soll nun u mal ausgeführt werden, so daß die Fehlerwahrscheinlichkeit insgesamt kleiner als $2^{-\omega}$ bleibt, für vorgegebene u und ω.

Für diese u Ausführungen werden $u(\alpha+1) + \beta$ Verteilungen benötigt. Jede bedeutet eine Ausführung von Protokoll 4.3 mit einem Fehler $\leq 2^{-\sigma}$. Es ist daher der Gesamtfehler

$$\leq (u(\alpha+1) + \beta)\, 2^{-\sigma} \leq 2^{-\omega} \Leftarrow \sigma \geq \omega + \log_2(\alpha+\beta+1) + \log_2(u).$$

Damit genügt also ein in u nur logarithmisches σ. u geht in die echte Vorbereitung (d. h. die mittels zuverlässigem Verteilkanal) nur via σ ein, d. h. nur logarithmisch.

Man sieht insbesondere, daß $\sigma = O(\omega + \log(u) + \log(n))$. Damit kann man auch die Gesamtrundenanzahl der u Ausführungen abschätzen. Sie ist

$$O(\sigma(u + n^2)) = O((\omega + \log(u) + \log(n))\,(u+n^2)).$$

Nimmt man an, u sei polynomial in n mit $u(n) = n^c$ für ein $c \geq 2$, so erhält man insgesamt $O((\omega + \log(n))u(n))$ Runden.

4.4 Zusammenfassung

In diesem Kapitel wurde ein byzantinisches Verteilungsprotokoll für Nachrichten aus $\{0,1\}^z$ vorgestellt, das jedes $t < n/2$ toleriert. Das Protokoll ist polynomial in n, z und einem Sicherheitsparameter σ und hat eine Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$.

Das Protokoll basiert auf einem authentisierten Protokoll von TOUEG, das für $t < n/2$ die byzantinische Verteilung langer Nachrichten effizient in 5 Runden

auf binäre byzantinische Übereinstimmung reduziert.

In diesem Protokoll wurden die digitalen Signaturen durch die in Abschnitt 3.1 eingeführten Pseudosignaturen ersetzt. In Kombination mit Protokoll 2.9 aus Kapitel 2 terminiert es in erwartet 23 Runden, im schlimmsten Fall in $O(\sigma)$ Runden.

Indem man das Verteilungsprotokoll selbst dazu verwendet, die für seine nachfolgenden Ausführungen notwendige zuverlässige Verteilung von Nachrichten zu realisieren, erhält man ein Verteilungsprotokoll, das mit einer in u logarithmischen Vorbereitung u mal ausgeführt werden kann. Die u Verteilungen benötigen insgesamt $O((\omega + \log(u) + \log(n))\ (u+n^2))$ Runden, wobei die Fehlerwahrscheinlichkeit insgesamt $\leq 2^{-\omega}$ ist.

5 Anwendung auf Mehrparteienberechnungen

In diesem Kapitel wird skizziert, welche Auswirkung die Verteilungsprotokolle der letzten Kapitel auf das Problem der fehlertoleranten und datenschützenden Mehrparteienberechnung einer beliebigen Funktion haben.[37]

5.1 Definitionen und Fakten

Eine **Mehrparteienberechnung** (*multi-party computation*) ist die datenschützende Auswertung einer Funktion f, deren Argumente von verschiedenen Prozessoren stammen und deren Ergebnis allen Prozessoren bekannt werden muß [Yao_82, GoMW_87] (auch: [Beav_90, BeGo_89, BeGW_88, ChCD_88, ChDG_88, GaHY_88, GoVa_88, Kush_89, RaBe_89, Yao_86]).

Es werden nur endliche Funktionen, d. h. solche mit endlichen Definitions- und Wertemengen, betrachtet. O. B. d. A. werden sowohl die Argumente als auch das Ergebnis aus einem endlichen Körper F gewählt und f als arithmetischer Ausdruck über F dargestellt.[38] Die Argumente entsprechen dann den Variablen in f. Die Zuordnung der Argumente von f zu den n Prozessoren erfolgt statisch.

Betrachtet werden hier nur *deterministische* Berechnungen. Zufallsentscheidungen können jedoch wie in Lemma 2.7 als Summe von t+1 zufälligen Eingaben von t+1 Prozessoren berechnet werden.

Sei wieder t die maximal erlaubte Anzahl an Störern. Im byzantinischen Fehlermodell stellt eine ***t*-sichere Berechnung** folgendes sicher (evtl. mit einer exponentiell kleinen Fehlerwahrscheinlichkeit):

Zur leichteren Notation seien die Argumente der korrekten Prozessoren zu x, die der Störer zu y zusammengefaßt, und zu berechnen sei $f(x, y)$.

Fehlertoleranz Alle Prozessoren erhalten nach begrenzter Zeit ein und dasselbe Ergebnis $z \in F$.

Korrektheit Die Definition der Korrektheit ist etwas umständlich, da die Störer sich nicht an das Protokoll halten und ihre Eingaben y daher nicht existieren müssen.

---

[37] Auf *spezielle* Mehrparteienberechnungen können sich die hier vorgestellten Hilfsmittel evtl. mit noch größerem Erfolg anwenden lassen, etwa auf die Probleme der unbeobachtbaren Kommunikation und geheimen Wahl [PfWa1_92].

[38] Ist etwa F = GF(2), so erhält man gerade die BOOLEschen Schaltnetze mit den Operatoren „und“ ($x \wedge y \cong x\,y \bmod 2$) und „antivalent“ ($x \leftrightarrow y \cong x+y \bmod 2$).

Ist x festgelegt, so definiert x und das probabilistische Verhalten aller Prozessoren eine Wahrscheinlichkeitsverteilung auf den möglichen Ergebnissen z.

Es wird gefordert: Diese Verteilung soll ununterscheidbar sein von einer Verteilung, die entsteht, wenn die Störer gemäß einer beliebigen, aber von x unabhängig gewählten Wahrscheinlichkeitsverteilung ein y^* wählen und z gemäß Protokoll als $f(x, y^*)$ berechnet werden würde.

Konzelation Die Störer erhalten über die Eingaben x der korrekten Prozessoren keine Information außer $z = f(x, y^*)$ für ein beliebiges, aber *unabhängig* von x gewähltes y^*.

Die Formulierung der drei Forderungen ist am byzantinischen Fehlfunktionsmodell orientiert. Im kryptographischen Fehlfunktionsmodell sehen die Forderungen etwas anders aus [GoMW_87], und häufig wird die t-sichere Berechnung dort auch ganz anders definiert.

Ein ***t*-sicheres Mehrparteienberechnungsprotokoll** ist ein Protokoll, das *jede* Funktion f mit endlichen Ein- und Ausgabebereichen t-sicher berechnen kann.

Beispiel: Eine einfache Mehrparteienberechnung ist die Berechnung der Funktion

$$\Sigma(r_1, \ldots, r_n) := r_1 + \ldots + r_n,$$

wo $r_i \in \mathrm{GF}(2^u)$, u beliebig, von Prozessor P_i beliebig gewählt wird. Diese Funktion wird durch das DC-Protokoll (Protokoll 3.1) berechnet (Lemma 3.1a).

Ersetzt man lokale Summen, die nicht verteilt werden, durch 0, und geht von einem zuverlässigen Verteilkanal aus, so ist das DC-Protokoll *fehlertolerant*, denn alle korrekten Prozessoren berechnen dann dieselbe globale Summe.

Das DC-Protokoll ist im obigen Sinne aber *nicht korrekt*: Wegen der hiesigen Rundendefinition können die Störer ihre Eingaben y wählen, *nachdem* sie die Summe der Eingaben x der korrekten Prozessoren erfahren haben. Damit können sie z. B. deterministisch das Ergebnis $z = \Sigma(x, y) = 0$ erzwingen. Durch ein unabhängig von x verteiltes y^* könnte ein solches Ergebnis „z konstant 0“ jedoch nicht erreicht werden.

Die *Konzelation* ist gesichert, denn die Störer erhalten unabhängig von ihrem Verhalten nur die Summe der Eingaben x (Lemma 3.1b).

Unter Verwendung des Protokolls zur überprüften Geheimnisaufteilung aus Abschnitt 2.1 kann man die Korrektheit des DC-Protokolls für $t < n/2$ bis auf einen exponentiell kleinen Fehler „reparieren“:

Die DC-Runde wird in zwei Phasen aufgespalten. In der ersten teilt jeder Prozessor seine lokale Summe mittels Phase 1 von Protokoll 2.6 auf. Wird ein

Prozessor dabei disqualifiziert, so wird seine lokale Summe durch 0 ersetzt. In der zweiten Phase werden die lokalen Summen zusammengesetzt und hieraus die globale Summe berechnet.

Damit müssen auch die Störer ihre lokalen Summen wählen, bevor sie irgendwelche Information über die Eingaben x der korrekten Prozessoren haben. ✎

Alle bekannten im byzantinischen Fehlfunktionsmodell t-sicheren Mehrparteienberechnungsprotokolle gehen von einem zuverlässigen Verteilkanal und nichtabhörbaren sicheren Verbindungen zwischen den Prozessoren aus.

Im byzantinischen Fehlfunktionsmodell gibt es fehlerfreie t-sichere Mehrparteienberechnungsprotokolle dann und nur dann, wenn $t < n/3$ ([BeGW_88] bzw. [ChCD_88, DDWY_90]). In diesem Fall kann der zuverlässige Verteilkanal fehlerfrei durch byzantinische Verteilung ersetzt werden.

Für $t < n/2$ gibt es Lösungen mit einer beliebig (exponentiell) kleinen Fehlerwahrscheinlichkeit [Beav_90, RaBe_89]. Die Protokolle in [Beav_90, RaBe_89] basieren auf RABINs Protokoll zur überprüften Geheimnisaufteilung (Protokoll 2.4).[39]

Alle bekannten t-sicheren Mehrparteienberechnungsprotokolle erhöhen den Aufwand zur Berechnung von f um einen in n (und ggf. im Sicherheitsparameter γ) polynomialen Faktor. Für praktische Anwendungen sind sie im allgemeinen zu aufwendig.

5.2 Anwendung der byzantinischen Verteilungsprotokolle

Die in den Kapiteln 2 bis 4 beschriebenen byzantinischen Verteilungsprotokolle können dazu verwendet werden, nach einer geeigneten Vorbereitung einen zuverlässigen Verteilkanal zu implementieren. Dies ist möglich, denn die Rundenanzahl aller Verteilungsprotokolle ist deterministisch beschränkt, d.h. es kann deterministisch festgesetzt werden, wann eine Runde des Mehrparteienberechnungsprotokolls endet und die nächste beginnen kann (vgl. Abschnitt 4.3).

Damit kann man insbesondere die t-sicheren Mehrparteienberechnungsprotokolle aus [Beav_90, RaBe_89] für $t < n/2$ ohne zuverlässigen Verteilkanal im Hauptprotokoll realisieren: Überall, wo ein Prozessor eine Nachricht zuverlässig verteilen muß, wird das Verteilungsprotokoll verwendet.

Für die Verteilungsprotokolle ist notwendig, daß in einem Vorbereitungsprotokoll mit zuverlässigem Verteilkanal und nichtabhörbaren sicheren Verbindungen hinreichend viele Verteilungen vorbereitet werden. Dies kann zu einem beliebigen Zeitpunkt geschehen, also insbesondere bevor die Prozessoren

---

[39] In [BIBe_89] wurde für $n/3 \le t < n/2$ von der falschen Annahme ausgegangen, das Aufteilen eines Geheimnisses nach [RaBe_89, Rabi_88] benötige nur eine konstante Rundenanzahl.

ihre Eingaben für f kennen. Im Vorbereitungsprotokoll können auch wie üblich Schlüssel ausgetauscht werden, so daß für das eigentliche Protokoll die nichtabhörbaren sicheren Verbindungen durch nur sichere ersetzt werden können.

Damit gilt der folgende Satz:

Satz 5.1: Im byzantinischen Fehlermodell gibt es für $t < n/2$ ein t-sicheres Mehrparteienberechnungsprotokoll mit exponentiell kleiner Fehlerwahrscheinlichkeit. Das Protokoll benötigt für das Vorbereitungsprotokoll einen zuverlässigen Verteilkanal und nichtabhörbare sichere Verbindungen, für das eigentliche Mehrparteienberechnungsprotokoll nur sichere Verbindungen. ♦

6 Zusammenfassung

In dieser Arbeit wurde das Problem der byzantinischen Verteilung unter dem Gesichtspunkt untersucht, unter möglichst schwachen Voraussetzungen möglichst viele fehlerhafte Prozessoren, hier Störer genannt, tolerieren zu können.

Ausgegangen wurde von n Prozessoren, die synchron arbeiten und paarweise zuverlässig in bekannt begrenzter Zeit miteinander kommunizieren können. Von diesen n Prozessoren können einige fehlerhaft werden; der Parameter t gab für ein Protokoll jeweils die maximal zu tolerierende Anzahl an fehlerhaften Prozessoren an. Über das Verhalten fehlerhafter Prozessoren wurden *keine* Annahmen gemacht, d.h. sie können sich im schlimmsten Fall koordiniert und planvoll verhalten und unterliegen auch keinen Aufwandsbeschränkungen. Dies ist das übliche *byzantinische Fehlfunktionsmodell*.

In diesem byzantinischen Fehlermodell waren bislang nur Protokolle bekannt, die jede Anzahl $t < n/3$ tolerieren können.

Unter der Annahme, ein geeigneter Authentikationsmechanismus stünde zur Verfügung, kann jedes $t < n$ toleriert werden (*authentisierte Protokolle*). Die einzigen bislang bekannten geeigneten Authentikationsmechanismen waren *digitale Signatursysteme*, und diese sind im byzantinischen Fehlfunktionsmodell prinzipiell unsicher.

Die hier vorgestellten Verteilungsprotokolle sind damit die ersten, die jede Anzahl $t < n/2$ bzw. sogar $t < n$ im byzantinischen Fehlermodell tolerieren.

Die Verteilungsprotokolle finden jeweils mit einer *Fehlerwahrscheinlichkeit* $\varepsilon > 0$ *keine* korrekte Lösung. Nach [PeSL_80] ist dies jedoch bei Tolerierung einer Anzahl $t \geq n/3$ fehlerhafter Prozessoren unvermeidbar. Die Fehlerwahrscheinlichkeit ε ist hier stets exponentiell klein, d.h. es gibt einen frei wählbaren Sicherheitsparameter σ, so daß die Fehlerwahrscheinlichkeit $\varepsilon \leq 2^{-\sigma}$ ist, während der Aufwand des jeweils betrachteten Protokolls polynomial in n und σ ist.

Bevor eines der Verteilungsprotokolle ausgeführt werden kann, müssen die Prozessoren in einem *Vorbereitungsprotokoll* gewisse Informationen *nicht-abhörbar austauschen* und *zuverlässig verteilen*. Beide Kommunikationsmöglichkeiten während des Vorbereitungsprotokolls sind im gewählten Modell unverzichtbar für eine Fehlerwahrscheinlichkeit $\varepsilon < 1/3$ (vgl. Schluß von Abschnitt 1.3).

Das Vorbereitungsprotokoll kann zu einem beliebigen Zeitpunkt ausgeführt werden, insbesondere bevor der Wert des Transmitters, der verteilt werden soll, feststeht. Es entspricht daher in gewisser Weise der Schlüsselaustauschphase bei

digitalen Signaturen.

Mit diesen Voraussetzungen wurde in Kapitel 2 ein Protokoll vorgestellt, das mit exponentiell kleiner Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ jede Anzahl $t < n/2$ an Störern toleriert (Protokoll 2.9).

Es terminiert in erwartet 19, im schlimmsten Fall in $O(\sigma)$ Runden. Das Protokoll kann nur Binärnachrichten (also solche aus {0,1}) verteilen. Durch ein authentisiertes Protokoll von TOUEG [Toue_84], in das ein in Kapitel 3 vorgestellter Authentikationsmechanismus eingesetzt wurde, wurde in Kapitel 4 jedoch die Verteilung langer Nachrichten effizient auf die Verteilung binärer Nachrichten reduziert.

Grundlage des Protokolls in Kapitel 2 ist ein Verfahren zur überprüften Geheimnisaufteilung (*verifiable secret sharing*) für $t < n/2$ im byzantinischen Fehlfunktionsmodell. Das einzige bislang bekannte solche Verfahren war das von RABIN [RaBe_89], das in seiner ursprünglichen Form jedoch nicht verwendbar war. Es konnte aber in dieser Arbeit verbessert und so modifiziert werden, daß nur noch zum Aufteilen, nicht aber zum Zusammensetzen eines Geheimnisses ein zuverlässiger Verteilkanal benötigt wird. Dies ermöglicht es, im eigentlichen Verteilungsprotokoll Geheimnisse zusammenzusetzen, die zuvor im Vorbereitungsprotokoll aufgeteilt wurden.

In Kapitel 3 wurde ein neuer Authentikationsmechanismus, die *Pseudosignaturen*, vorgestellt und in das authentisierte Verteilungsprotokoll von DOLEV und STRONG statt üblicher digitaler Signaturen eingesetzt [DoSt_83]. Das Resultat war das erste Verteilungsprotokoll, das mit exponentiell kleiner Fehlerwahrscheinlichkeit $\leq 2^{-\sigma}$ jede Anzahl $t < n$ an Störern toleriert (Protokoll 3.6).

Es terminiert (wie das ursprüngliche Protokoll von DOLEV und STRONG) in deterministisch $t+1$ Runden, im Gegensatz zu den erwartet O(1) Runden des obigen Protokolls für $t < n/2$.

Pseudosignaturen basieren im Gegensatz zu echten digitalen Signaturen auf keinen kryptographischen Annahmen. Erhält ein Prozessor eine Pseudosignatur ψ, so lehnt er sie entweder als ungültig ab oder ordnet ihr eine Akzeptionsstufe $k \in \{1, \ldots, \lambda\}$ (für einen festen Parameter λ) zu. k-akzeptiert er ψ, so ist ψ echt, und ist zusätzlich $k < \lambda$, so wird jeder andere korrekte Prozessor sie wenigstens $(k+1)$-akzeptieren (beides jeweils bis auf exponentiell kleine Fehler). Im Gegensatz zu digitalen Signaturen können Pseudosignaturen also nur endlich, nämlich λ mal sequentiell sicher weitergegeben werden. Pseudosignaturen verallgemeinern ein Verfahren von CHAUM und ROIJAKKERS, das speziell den Fall $\lambda = 2$ betrachtete [ChRo_91].

Die Sicherheit von Pseudosignaturen hängt wesentlich vom Protokoll ab, in dem sie eingesetzt werden. Es zeigte sich, daß mit genügend großem Aufwand Pseudosignaturen für eine Vielzahl authentisierter Protokolle ein sichererer

Ersatz für nur kryptographisch sichere digitale Signaturen sein können.

In Kapitel 5 wurde als interessante theoretische Anwendung der vorgestellten Verteilungsprotokolle das Problem der t-sicheren Mehrparteienberechnungen beschrieben. Dies ist das Problem, eine Funktion f verteilt und fehlertolerant so zu berechnen, daß außer dem Funktionsergebnis nichts über die Eingaben der korrekten Prozessoren bekannt wird, vorausgesetzt, es gibt höchstens t Störer. Bislang war bekannt, daß für $t < n/2$ in einem zuverlässigen Verteilnetz jede Funktion t-sicher berechnet werden kann [Beav_90, RaBe_89]. Setzt man die byzantinischen Verteilungsprotokolle als Ersatz für das vorausgesetzte zuverlässige Verteilnetz ein, so gilt diese Aussage nun auch in Punkt-zu-Punkt-Netzen, falls zuvor das Vorbereitungsprotokoll des verwendeten Verteilungsprotokolls hinreichend oft ausgeführt wurde.

Danksagung

Mein herzlicher Dank geht an alle, die meine Arbeit unterstützt haben:

Prof. Dr.-Ing. *Winfried Görke* danke ich für die Übernahme des Referats und der Betreuung meiner Arbeit, und für die wissenschaftliche und organisatorische Freiheit und Unterstützung, die er mir in den letzten Jahren an seinem Lehrstuhl gewährte. Prof. Dr. *Rüdiger Reischuk* danke ich für die freundliche und frühzeitige Übernahme des Korreferats und für einige anregende Diskussionen.

Für anregende Diskussionen danke ich auch *Manfred Böttger*, *Winfried Bücken*, Prof. Dr. *Klaus Echtle*, *Kristin Lucie Munthe* und *Andreas Ort*. Dr. *David Chaum* danke ich für die frühzeitige Überlassung seiner und *Sandra Roijakkers* Arbeit [ChRo_91]. Viel Unterstützung erfuhr ich von Dr. *Andreas Pfitzmann*. Inhaltlich verdanke ich ihm den Hinweis auf die Möglichkeit aktiver Angriffe auf Pseudosignaturen (vgl. Abschnitt 3.1.6).

Besonderen Dank schulde ich meiner Ehepartnerin *Birgit Baum-Waidner*, von deren Kenntnissen über byzantinische Verteilung ich sehr profitierte, und ohne deren menschliche und organisatorische Unterstützung diese Arbeit sicher nicht zustande gekommen wäre. Die Protokolle aus Kapitel 2 mußten stets erst ihrer Kritik standhalten, bevor sie als sicher und richtig gelten durften. Den Inhalt eines Großteils von Kapitel 2 veröffentlichte ich zusammen mit ihr und *Birgit Pfitzmann* in [BaPW_91].

Mit *Birgit Pfitzmann* verbinden mich so viele gemeinsame Arbeiten und Ideen, daß es fast unmöglich ist, mich bei ihr im einzelnen zu bedanken. Ihrer gründlichen Durchsicht der Arbeit verdanke ich zahlreiche Verbesserungsvorschläge. Den Inhalt eines Großteils von Kapitel 3 veröffentlichte ich zusammen mit ihr in [PfWa_92]. Ich wünsche ihr, daß sie und ihre Arbeit ebenso von mir profitieren werden, wie dies umgekehrt der Fall war.

Häufig verwendete Symbole und Abkürzungen

α_i	Stützstelle für P_i 34; 37
$Auth$	Codieralgorithmus für Authentikationscode 81
$\alpha(z,\tau)$	Länge eines Authentikationsschlüssels 82
(b, c)	Prüfpaar 37
B_h	Binomialverteilung 102
β_i	Anteil von P_i an einem Geheimnis 32
C_i	lokale Uhr von P_i 22
Δ	Akzeptionsdifferenz einer Pseudosignatur 99
Δ_C	Abweichung lokaler Uhren 22
Δ_D	Drift lokaler Uhren 22
Δ_P	Verarbeitungsdauer 23
Δ_T	Transferdauer 22
E	Empfängerprozessor 35
$\varepsilon(R)$	Fehlerwahrscheinlichkeit nach R Runden 18; 24
F	Endlicher Körper 30
$F^{(k)}$	Endlicher Körper 36
$F[X]$	Menge aller Polynome über F 30
γ	Sicherheitsparameter für Mehrfachzugriff 85
G	Geheimniskrämer 33
$Gener$	Algorithmus zur Schlüsselgenerierung für ein digitales Signatursystem 9
GF(p)	Endlicher Körper mit p Elementen 30
$g(n,\omega)$	Anzahl der Pseudosignaturen eines ordentlichen Protokolls 104
$h(n,\omega)$	Anzahl der Tests einer Pseudosignatur je Prozessor in einem ordentlichen Protokoll 105
ι	Codierung von {0, …, L–1} in F 60
k	kryptographischer Sicherheitsparameter 10
k	Sicherheitsparameter für Geheimnisaufteilung 36; 58
κ	Sicherheitsparameter für Pseudosignaturen 99
K	Anzahl der Iterationen von Protokoll 2.7 in Protokoll 2.8 66
$K_{i,j}$	DC-Schlüssel von P_i und P_j 83
L	Sicherheitsparameter für byz. Verteilung 59
λ	Akzeptionsstufen einer Pseudosignatur 78
m	Sicherheitsparameter für Geheimnisaufteilung 45
m	Vielfachheit einer Pseudosignatur 79; 99
$\|M\|$	Anzahl der Elemente der Menge M
n	Anzahl aller Prozessoren 4
N	Nachricht 8; 13
O_i	Lokale Summe von P_i 83

P, Q, R	Prozessoren 13
P_i	i-ter Prozessor, $i \in \{1, \ldots, n\}$ 13
R	Münzwurfergebnis 60
R	Rundenanzahl 17; 24
r^*	$= \iota(r)$ 60
r_i	Zufällig aus $\{0, \ldots, L–1\}$ gewählter Wert 60
R_u	Münzwurf in Schritt $[u]$ 71
s	Geheimnis für Geheimnisaufteilung 31
σ	Sicherheitsparameter für byz. Verteilung 18; 66
S	Signierer 78
Σ	Globale Summe 83
Σ_h	Potenzsumme ${x_1}^h + \ldots + {x_k}^h$ 85
s_i	Geheimnis von P_i 60
$Sign$	Signieralgorithmus 9
SSi	Signierschlüssel 9; 79; 99
STe	Testschlüssel für (Pseudo-)signaturen 9
STe_i	Testschlüssel von P_i 79; 99
$Test$	Testalgorithmus für Signaturen 9
$T_{i,j}$	j-ter Testschlüssel von P_i 79
t	Maximal zulässige Anzahl an Störern 5
t	Schwellwert für Geheimnisaufteilung 31
τ	Sicherheitsparameter für Authentikationscode 81
τ	Sicherheitsparameter für Geheimnisaufteilung 32; 58
T	Transmitter 60
u	Anzahl der Ausführungen eines Protokolls 29; 33; 52
u	Parameter für DC-Protokoll 83
u_k	$= \lceil k + \log_2(k) + 1 \rceil$ 36
v	Zu verteilende Nachricht 4
${v_i}^{(l)}$	Wert von P_i in der l-ten Iteration, $l \in \{0, \ldots, L–1\}$ 61
$v_{i,u}$	Wert von P_i in Schritt $[u]$ 67
$W(A)$	Wahrscheinlichkeit für Ereignis A 18
ω	Sicherheitsparameter für byz. Pseudoverteilung 125
ω	Sicherheitsparameter für ordentliches Protokoll 104
y	Authentikationsnachweis 37
ψ	Pseudosignatur 80; 99
ψ_j	j-ter Abschnitt von ψ, $j \in \{1, \ldots, m\}$ 80; 99
z	Nachrichtenlänge in bit 81
Z	Zwischenprozessor 35
Π_t	Menge aller Polynome vom Grade $\leq t$ 30
$\Pi_t(s)$	Menge aller Polynome f in Π_t mit $f(0) = s$ 30

Stichwörter

α-Signatur 109
ABELsche Gruppe 84
Abschnitte
  einer Pseudosignatur 80; 99
Akzeptionsstufen
  einer Pseudosignatur 80; 100
Anfangswert
  des Transmitters 4
Angriffe
  aktive, auf Pseudosignaturen 81; 103
  passive, auf Pseudosignaturen 101
Anhalteausfall 12
Annahme
  kryptographische 11
Anteil
  eines Geheimnisses 31
Aufdecken
  im DC-Protokoll 93
Aufteilen
  eines Geheimnisses 31
Aufwand
  eines Protokolls 18
  erwarteter 18
  maximaler 18
Auswahl
  zufällige 30
Authentikationscode 8; 25; 81
Authentikationsmechanismus 7; 78
Authentikationsnachweis 8; 32; 35; 37; 39
Authentikationsschlüssel
  eines Authentikationscodes 81
Authentizität
  einer sicheren Verbindung 16
  einer zuverlässigen Verteilung 17
Berechnungsprotokoll
  Grenzen 138
  t-sicheres 137
Beschwerdenachricht
  im Initialisierungsprotokoll einer Pseudosignatur 90
Binomialverteilung 102
BLUM-Zahlen 11
BOOLEsches Schaltnetz 136
BOS
  Mehrfachzugriffsprotokoll von … 84
broadcast primitive
  als Authentikationsmechanismus 8
Byzantine agreement
  s. byzantinische Verteilung 4
Byzantine generals problem
  s. byzantinische Verteilung 4
CHAUM
  DC-Protokoll von … 82
  Pseudosignaturen von … 78; 79
consensus problem
  s. Übereinstimmungsproblem 5
crash 12
crusader agreement
  s. byzantinische Pseudoverteilung 124
cut-and-choose 38; 44; 45
DC-
  Nachricht 83
  Protokoll 82; 83; 84; 87
  Runde 83
  Schlüssel 83
dealer
  s. Geheimniskrämer 31
DEN BOER
  Mehrfachzugriffsprotokoll von … 84
dining cryptographers
  s. DC-Protokoll 82
Disqualifizierung
  öffentliche 33; 92; 93
distributed firing squad problem 134
DOLEV
  Verteilungsprotokoll von … 78; 109
Endwert
  eines Prozessors 4
Erneuerungsprotokoll 118; 122
eventual Byzantine agreement 70
fail-stop 12
fast sicher 19
fast unmöglich 19
Fehler
  byzantinischer 5
  Terminierungsfehler 24
Fehlerlokalisierung
  im DC-Protokoll 93
Fehlermodell 15
  byzantinisches 15; 17
Fehlersituation 15
  zulässige 15
Fehlertoleranz 3; 5; 17
  einer t-sicheren Berechnung 137
Fehlerwahrscheinlichkeit
  beliebig/exponentiell kleine 9
  eines Protokolls 18
  für Terminierung 25
  reale 27
Fehlfunktionsmodell 7
  Anhaltemodell 12
  byzantinisches 5; 15

kryptographisches 10
Geheimnis
für Geheimnisaufteilung 31
öffentliches 38
Geheimnisaufteilung
einfache 31; 34
Grenzen 31
modifizierte überprüfte 56
überprüfte 30; 31; 43; 45; 47
Grenzen 31
Geheimnisfestlegung 40; 41
schwache 32; 54
Geheimniskrämer
für Geheimnisaufteilung 31
Grundinitialisierungsprotokoll
einer Pseudosignatur 82; 89; 90
information checking
s. Authentikationsnachweis 8
Initialisierungsprotokoll
einer Pseudosignatur 80; 82; 88; 92; 94
Integrität
einer sicheren Verbindung 16
interactive consistency
s. interaktive Übereinstimmung 5
KARLIN
Resultat von … 21
Konsistenz
der byzantinischen Pseudoverteilung 125
der byzantinischen Verteilung 4
einer zuverlässigen Verteilung 17
Konzelation
einer *t*-sicheren Berechnung 137
sichere nichtabhörbare Verbindung 16
Körper
endlicher 30
Korrektheit
der byzantinischen Pseudoverteilung 125
der byzantinischen Verteilung 4
einer Authentikation 7
einer digitalen Signatur 10
einer *t*-sicheren Berechnung 137
einer zuverlässigen Verteilung 17
von Pseudosignaturen 101; 105
kryptographisch sicher 10
LAGRANGE
Interpolation nach … 34
Mehrfachzugriff
für DC-Protokoll 85; 87
Mehrparteienberechnung 136
Minisignaturen
in einer Pseudosignatur 80; 81
multi-party computation
s. Mehrparteienberechnung 136
Münzwurf 59; 62
Nachrichtenaufwand 18
NEWTON
Gleichungen von … 86
one-time-pad
s. VERNAM-CHIFFRE 25
Protokoll 13
authentisiertes 7; 104
deterministisches 13
ordentliches 104; 115
probabilistisches 13; 14
Prozessorprotokoll 13
Protokollnachricht
k-konsistente 110
Prozessor 4
Modell 13
Prüfpaar
für Authentikationsnachweis 37
Pseudosignaturen 80
einfache 9; 80
mit λ Akzeptionsstufen 78; 99
Pseudoverteilung
byzantinische 124; 125; 126; 127
public key
s. Signierschlüssel 10
Punkt-zu-Punkt-Netz 4
RABIN, MICHAEL
Faktorisierungsalgorithmus von … 87
RABIN, TAL
Aufteilungsprotokolle von … 30
Authentikationsnachweis von … 35
Geheimnisfestlegungsprotokoll von … 40
Rechenaufwand 18
Rekonfigurierung
im DC-Protokoll 94
reliable broadcast
s. byzantinische Verteilung 4
ROIJAKKERS
Pseudosignaturen von … 78; 79
Runde 13
Rundenanzahl 18
Schwellwert
für Geheimnisaufteilung 31
secret key
s. Testschlüssel 10
secret sharing
s. einfache Geheimnisaufteilung 31
SHAMIR
einfache Geheimnisaufteilung von … 34
SHANNON
Informationsbegriff von … 31
Sicherheitsparameter
allgemein 18
kryptographischer 10
Signaturen
digitale 9; 38; 79; 100
Fail-stop 12; 27

Signierschlüssel
einer Pseudosignatur 94; 99
für digitale Signaturen 10
simultaneous Byzantine agreement 70
Störer 5; 15
Störprotokoll 15
o. B. d. A. deterministisch 106; 107
Strategiebaum
für Störprotokoll 108; 116
STRONG
Verteilungsprotokoll von … 78; 109
Summe
globale, im DC-Protokoll 83
lokale, im DC-Protokoll 83
synchron
Kommunikation 13; 22
Prozessoren 13
Uhren 17; 22
Terminierung
der byzantinischen Pseudoverteilung 125
der byzantinischen Verteilung 4
einer zuverlässigen Verteilung 17
eines Protokolls 13
Testschlüssel
einer Pseudosignatur 80; 94; 99
für digitale Signaturen 10
TOUEG
Filterprotokoll von … 128
Verteilungsprotokoll von … 124; 130
Transmitter 4
TURING-Maschine
interaktive 14
probabilistische 14
Übereinstimmung
interaktive 5
Übereinstimmungsproblem 5; 132
Uhrensynchronisation
byzantinische 24
unconditionally secure digital signatures
s. Pseudosignaturen 9
Unfälschbarkeit
einer Authentikation 7
einer digitalen Signatur 10
von Pseudosignaturen 101; 105
Verbindung
sichere 16
Realisierung 25
sichere nichtabhörbare 16
verifiable secret sharing
s. überprüfte Geheimnisaufteilung 30
VERNAM-Chiffre 25
Veröffentlichen
einer Nachricht 33
Verteilaufwand 18
Verteilkanal
zuverlässiger 16
Verteilung
asynchrone byzantinische 24
byzantinische
Definition 4
für $t < n$
mit exp. kleinem Fehler 113
für $t < n/2$
mit exp. kl. Fehler 66; 67
mit exp. kl. Fehler in erwartet 19 Runden 70; 72
mit linear kl. Fehler 60; 62
Grenzen 19
zuverlässige 16
Verteilungsprotokoll
authentisiertes 7
binäres 4; 128
byzantinisches 4
kombiniert sicheres 11
probabilistisches 20
Vetonachricht
für Geheimnisfestlegung 41
Vielfachheit
einer Pseudosignatur 79; 92; 99
Vorbereitungsprotokoll 16
für digitale Signaturen 10
Realisierung 26
weak secret sharing
s. Geheimnisfestlegung 40
Weitergebbarkeit
einer Authentikation 7
einer digitalen Signatur 10
einer Pseudosignatur 78; 101; 107
Weiterleiten
einer Protokollnachricht 110
YAO
Resultat von … 21
Zufallsband
einer TURING-Maschine 14
Zusammenhang
bzgl. DC-Schlüssel 84; 92
Zusammenhangsannahme 16; 25
Zusammensetzen
eines Geheimnisses 31
Zustand
akzeptierender 13
eines Prozessors 14
Startzustand 13
Stopzustand 13
Zustandsraum 13

Literatur

Baba_87 Özalp Babaoglu: On the Reliability of Consensus-Based Fault-Tolerant Distributed Computing Systems; ACM Transactions on Computer Systems 5/4 (1987) 394-416.

BaPW_91 Birgit Baum-Waidner, Birgit Pfitzmann, Michael Waidner: Unconditional Byzantine Agreement with Good Majority; 8th Symposium on Theoretical Aspects of Computer Science (STACS) 1991, LNCS 480, Springer-Verlag, Berlin 1991, 285-295.

Baum_91 Birgit Baum-Waidner: Adaptive Byzantine Agreement in O(t) Phases; Fault-Tolerant Computing Systems 1991; IFB 283, Springer, Berlin 1991, 112-123.

Baum1_91 Birgit Baum-Waidner: A Consistency Condition Supporting Design and Verification of Byzantine Adaptive Agreement Protocols; Kommunikation in verteilten Systemen 1991, IFB 267, Springer-Verlag, Berlin 1991, 451-463.

Beav_90 Donald Beaver: Multiparty protocols tolerating half faulty processors; Crypto '89, LNCS 435, Springer-Verlag, Berlin 1990, 560-572.

BeGo_89 Donald Beaver, Shafi Goldwasser: Multiparty Computation with Faulty Majority; 30th Symposium on Foundation of Computer Science (FOCS) 1989, IEEE Computer Society, 1989, 468-473.

BeGW_88 Michael Ben-Or, Shafi Goldwasser, Avi Wigderson: Completeness theorems for non-cryptographic fault-tolerant distributed computation; 20th Symposium on Theory of Computing (STOC) 1988, ACM, New York 1988, 1-10.

BeKP_86 Richard Berger, Sampath Kannan, Rene Peralta: A Framework for the Study of Cryptographic Protocols; Crypto '85, LNCS 218, Springer-Verlag, Berlin 1986, 87-103.

BenO_83 Michael Ben-Or: Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols; 2nd Symposium on Principles of Distributed Computing (PODC) 1982, ACM, New York 1982, 89-103.

BIBe_89 Judit Bar-Ilan, Donald Beaver: Non-Cryptographic Fault-Tolerant Computing in a Constant Number of Rounds of Interactions; 8th Symposium on Principles of Distributed Computing (PODC) 1989, ACM , New York 1983, 201-209.

Blak_79 G. R. Blakley: Safeguarding cryptographic keys; AFIPS Conference Proceedings Band 48, National Computer Conference (NCC) 1979, 313-317.

BlPW_91 Gerrit Bleumer, Birgit Pfitzmann, Michael Waidner: A Remark on a Signature Scheme where Forgery can be Proved; Eurocrypt '90, LNCS 473, Springer-Verlag, Berlin 1991, 441-445.

BoBo_90 Jurjen Bos, Bert den Boer: Detection of disrupters in the DC protocol; Eurocrypt '89, LNCS 434, Springer-Verlag, Berlin 1990, 320-327.

Brac_85 Gabriel Bracha: An O(log n) expected rounds randomized Byzantine Generals protocol; 17th Symposium on Theory of Computing (STOC) 1985, ACM, New York 1985, 316-326.

Brac_87 Gabriel Bracha: An asynchronous [(n–1)/3]–resilient consensus Protocol; Information and Computation; 75/2 (1987) 130-144.

BrBr_88 Gilles Brassard, Paul Bratley: Algorithms; Prentice-Hall, Englewood Cliffs 1988.

BrCC_88 Gilles Brassard, David Chaum, Claude Crépeau: Minimum Disclosure Proofs of Knowledge; Journal of Computer and System Sciences 37 (1988) 156-189.

BrDo_84 Andrei Z. Broder, Danny Dolev: Flipping coins in many pockets; 25th Symposium on Foundations of Computer Science (FOCS) 1984, IEEE Computer Society, 1984, 157-170.

BrTo_85 Gabriel Bracha, Sam Toueg: Asynchronous Consensus and Broadcast Protocols; Journal of the ACM 32/4 (1985) 824-840.

Burt_71 Herbert O. Burton: Inversionless Decoding of Binary BCH Codes; IEEE Transactions on Information Theory 17/4 (1971) 464-466.

CDDS_85 Brian A. Coan, Danny Dolev, Cynthia Dwork, Larry Stockmeyer: The Distributed Firing Squad Problem (Preliminary Version); 17th Symposium on Theory of Computing (STOC) 1985, ACM, New York 1985, 335-345.

CGMA_85 Benny Chor, Shafi Goldwasser, Silvio Micali, Baruch Awerbuch: Verifiable secret sharing and achieving simultaneity in the presence of faults; 26th Symposium on Foundations of Computer Science (FOCS) 1985, IEEE Computer Society, 1985, 383-395.

Chau_85 David Chaum: Security without Identification: Transaction Systems to make Big Brother Obsolete; Communications of the ACM 28/10 (1985) 1030-1044. Übersetzung: Sicherheit ohne Identifizierung; Scheckkartencomputer, die den Großen Bruder der Vergangenheit angehören lassen; Informatik-Spektrum 10/5 (1987) 262-277; Datenschutz und Datensicherung DuD /1 (1988) 26-41.

Chau_88 David Chaum: The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability; Journal of Cryptology 1/1 (1988) 65-75.

ChCD_88 David Chaum, Claude Crépeau, Ivan Damgård: Multiparty unconditional secure protocols; 20th Symp. on Theory of Computing (STOC) 1988, ACM, New York 1988, 11-19.

ChDG_88 David Chaum, Ivan B. Damgård, Jeroen van de Graaf: Multiparty Computations ensuring privacy of each party's input and correctness of the result; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 87-119.

ChDw_89 Benny Chor, Cynthia Dwork: Randomization in Byzantine Agreement; JAI Press, Advances in Computing Research 5, Greenwich 1989, 443-497.

ChRo_91 David Chaum, Sandra Roijakkers: Unconditionally Secure Digital Signatures; Crypto '90, LNCS 537, Springer-Verlag, Berlin 1991, 206-214.

DDWY_90 Danny Dolev, Cynthia Dwork, Orli Waarts, Moti Yung: Perfectly Secure Message Transmission; 31th Symposium on Foundations of Computer Science (FOCS) 1990, IEEE Computer Society, 1990, 36-45.

DeLM_82 Richard A. DeMillo, Nancy A. Lynch, Michael J. Merritt: Cryptographic Protocols; 14th Symposium on Theory of Computing (STOC) 1982, ACM, New York 1982, 383-400.

DFFL_82 Danny Dolev, Michael. J. Fischer, Rob Fowler, Nancy A. Lynch, H. Raymond Strong: An Efficient Algorithm for Byzantine Agreement without Authentication; Information and Control 52 (1982) 257-274.

DiHe_76 Whitfield Diffie, Martin E. Hellman: New Directions in Cryptography; IEEE Transactions on Information Theory 22/6 (1976) 644-654.

DoDw_91 Danny Dolev, Cynthia Dwork: On-The-Fly Generation of Names and Communication Primitives; IBM Research Division, Almaden Research Center, San Jose; erhalten am 13. 12. 1991 von Cynthia Dwork.

DoHS_84 Danny Dolev, Joe Halpern, H. Raymond Strong: On the Possibility and Impossibility of Achieving Clock Synchronization; 16th Symposium on Theory of Computing (STOC) 1984, ACM, New York 1984, 504-511.

Dole_82 Danny Dolev: The Byzantine Generals Strike Again; J. of Algorithms 3 (1990) 14-30.

DoRe_85 Danny Dolev, Rüdiger Reischuk: Bounds on Information Exchange for Byzantine Agreement; Journal of the ACM 32/1 (1985) 191-204.

DoRS_90 Danny Dolev, Rüdiger Reischuk, H. Raymond Strong: Early Stopping in Byzantine Agreement; Journal of the ACM 37/4 (1990) 720-741.

DoSt_83 Danny Dolev, H. Raymond Strong: Authenticated Algorithms for Byzantine Agreement; SIAM J. Comput. 12/4 (1983) 656-666.

DwMo_90 Cynthia Dwork, Yoram Moses: Knowledge and Common Knowledge in a Byzantine Environment: Crash Failures; Information and Computation 88 (1990) 156-186.

Echt_90 Klaus Echtle: Fehlertoleranzverfahren; Studienreihe Informatik, Springer-Verlag, Berlin 1990.

ErSp_74 Paul Erdös, Joel Spencer: Probabilistic Methods in Combinatorics; Probability and Mathematical Statistics 17, Academic Press, New York 1974.

FeMi_88 Paul Feldman, Silvio Micali: Optimal algorithms for byzantine agreement; 20th Symposium on Theory of Computing (STOC) 1988, ACM, New York 1988, 148-161.

FiLP_85 Michael J. Fischer, Nancy A. Lynch, Michael S. Paterson: Impossibility of Distributed Consensus with One Faulty Process; Journal of the ACM 32/2 (1985) 374-382.

FiLy_82 Michael J. Fischer, Nancy A. Lynch: A lower bound for the time to assure interactive consistency; Information Processing Letters 14/4 (1982) 183-186.

Fisc_83 Michael J. Fischer: The Consensus Problem in Unreliable Distributed Systems (A Brief Survey); 4th Conference on Foundations of Computation Theory, 1983, 127-140.

GaHY_88 Zvi Galil, Stuart Haber, Moti Yung: Cryptographic computation: secure fault-tolerant protocols and the public-key model; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 135-155.

Gill_77 John Gill: Computational Complexity of Probabilistic Turing Machines; SIAM J. Comput. 6/4 (1977) 675-695.

GiMS_74 E. N. Gilbert, F. J. Mac Williams, N. J. A. Sloane: Codes which detect deception; The Bell System Technical Journal 53/3 (1974) 405-424.

GoMR_88 Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM J. Comput. 17/2 (1988) 281-308.

GoMR_89 Shafi Goldwasser, Silvio Micali, Charles Rackoff: The Knowledge Complexity of Interactive Proof Systems; SIAM J. Comput. 18/1 (1989) 186-207.

GoMW_87 Oded Goldreich, Silvio Micali, Avi Wigderson: How to play any mental game - or - a completeness theorem for protocols with honest majority; 19th Symposium on Theory of Computing (STOC) 1987, ACM, New York 1987, 218-229.

GoMW_91 Oded Goldreich, Silvio Micali, Avi Wigderson: Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design; Journal of the ACM 38/1 (1991) 691-729.

GoPe_90 Oded Goldreich, Erez Petrank: The best of both Worlds: Guaranteeing Termination in fast Randomized Byzantine Agreement Protocols; Information Processing Letters 36/1 (1990) 45-49.

GoUd_61 Goscinny, Uderzo: Asterix der Gallier; Delta Verlag, Stuttgart 1990 (Neuauflage).

GoVa_88 Oded Goldreich, Ronen Vainish: How to solve any protocol problem - an efficiency improvement (extended abstract); Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 73-86.

GrYa_89 Ronald L. Graham, Andrew C. Yao: On the Improbability of Reaching Byzantine Agreements; 21st Symposium on Theory of Computing (STOC) 1989, ACM, New York 1989, 467-478.

Hadz_84 Vassos Hadzilacos: Issues of Fault Tolerance in Concurrent Computations; Ph.D. Thesis, Harvard University, Cambridge, Massachusetts, June 1984.

HePe_92 Eugène van Heijst, Torben Pryds Pedersen: How to make efficient Fail-stop signatures; Eurocrypt '92, Extended Abstracts, 24.-28. Mai 1992, Balatonfüred, Ungarn, 337-346.

HePP_92 Eugène van Heijst, Torben Pryds Pedersen: New Constructions of Fail-Stop Signatures and Lower Bounds; Crypto '92, Abstracts, August 1992, Santa Barbara, USA.

HSSD_84 Joseph Y. Halpern, Barbara Simons, Ray Strong, Danny Dolev: Fault-Tolerant Clock Synchronization; 3rd Symposium on Principles of Distributed Systems (PODC) 1984, ACM, New York 1984, 89-102.

Kush_89 Eyal Kushilevitz: Privacy and Communication Complexity; 30th Symposium on Foundation of Computer Science (FOCS) 1989, IEEE Computer Society, 1989, 416-421.

LaMe_84 Leslie Lamport, P. M. Melliar-Smith: Byzantine Clock Synchronization; 3rd Symposium on Principles of Distributed Systems (PODC) 1984, ACM, New York 1984. Reprinted in: Operating Systems Review 20/3 (1986) 10-16.

LaSP_82 Leslie Lamport, Robert Shostak, Marshall Pease: The Byzantine Generals Problem; ACM Transactions on Programming Languages and Systems 4/3 (1982) 382-401.

LiNi_86 Rudolf Lidl, Harald Niederreiter: Introduction to finite fields and their applications; Cambridge University Press, Cambridge (UK) 1986.

MoWa_88 Yoram Moses, Orli Waarts: Coordinated Traversal: (t+1)-Round Byzantine Agreement in Polynomial Time; 29th Symposium on Foundations of Computer Science (FOCS) 1988, IEEE Computer Society, 1988, 246-254.

PeSL_80 Marshall Pease, Robert Shostak, Leslie Lamport: Reaching Agreement in the Presence of Faults; Journal of the ACM 27/2 (1980) 228-234.

Pfit_89 Birgit Pfitzmann: Für den Unterzeichner unbedingt sichere digitale Signaturen und ihre Anwendung; Diplomarbeit am Institut für Rechnerentwurf und Fehlertoleranz der Universität Karlsruhe 1989.

Pfit_90 Andreas Pfitzmann: Diensteintegrierende Kommunikationsnetze mit teilnehmerüberprüfbarem Datenschutz; IFB 234, Springer-Verlag, Berlin 1990.

PfWa_90 Birgit Pfitzmann, Michael Waidner: Formal Aspects of Fail-stop Signatures; Interner Bericht 22/90 der Fakultät für Informatik, Universität Karlsruhe, Dezember 1990.

PfWa_91 Birgit Pfitzmann, Michael Waidner: Fail-stop-Signaturen und ihre Anwendung; Verläßliche Informationssysteme 1991, IFB 271, Springer-Verlag, Berlin 1991, 289-301.

PfWa1_91 Birgit Pfitzmann, Michael Waidner: Unbedingte Unbeobachtbarkeit mit kryptographischer Robustheit; Verläßliche Informationssysteme 1991, IFB 271, Springer-Verlag, Berlin 1991, 302-320.

PfWa2_91 Birgit Pfitzmann, Michael Waidner: Fail-stop Signatures and their Application; SECURICOM 91; 9th Worldwide Congress on Computer and Communications Security and Protection, 19.-22. March 1991.

PfWa_92 Birgit Pfitzmann, Michael Waidner: Unconditional Byzantine Agreement for any Number of Faulty Processors; 9th Symposium on Theoretical Aspects of Computer Science (STACS) 1992, LNCS 577, Springer-Verlag, Heidelberg 1992, 339-350.

PfWa1_92 Birgit Pfitzmann, Michael Waidner: Unconditionally Untraceable and Fault-tolerant Broadcast and Secret Ballot Election; Hildesheimer Informatik-Berichte (Mai 1992), Institut für Informatik, Universität Hildesheim.

RaBe_89 Tal Rabin, Michael Ben-Or: Verifiable Secret Sharing and Multiparty Protocols with Honest Majority; 21st Symposium on Theory of Computing (STOC) 1989, ACM, New York 1989, 73-85.

Rabi_78 Michael O. Rabin: Digitalized Signatures; Foundations of Secure Computation, ed. by R.A. DeMillo, D.P. Dobkin, A.K. Jones, R.J. Lipton; Academic Press, N.Y. 1978, 155-166.

Rabi_80 Michael O. Rabin: Probabilistic Algorithms in Finite Fields; SIAM J. Comput. 9/2 (1980) 273-280.

Rabi_83 Michael O. Rabin: Randomized Byzantine Generals; 24th Symposium on Foundations of Computer Science (FOCS) 1983, IEEE Computer Society, 1983, 403-409.

Rabi_88 Tal Rabin: Robust Sharing of Secrets when the Dealer is Honest or Cheating; Technical Report Nr. 1 1988, Computer Science Department, Hebrew University, Jerusalem, Israel.

Reis_87 Rüdiger Reischuk: Konsistenz und Fehlertoleranz in Verteilten Systemen - Das Problem der Byzantinischen Generäle; 17. GI Jahrestagung, IFB 156, Springer-Verlag, Berlin 1987, 65-81.

ScSc_83 Richard D. Schlichting, Fred B. Schneider: Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems; ACM Transactions on Computer Systems 1/3 (1983) 222-238.

Sham_79 Adi Shamir: How to Share a Secret; Communications of the ACM 22/11 (1979) 612-613.

Shan_49 Claude E. Shannon: Communication Theory of Secrecy Systems; The Bell System Technical Journal 28/4 (1949) 656-715.

SrTo_87 T. K. Srikanth, Sam Toueg: Simulating Authenticated Broadcasts to Derive Simple Fault-Tolerant Algorithms; Distributed Computing 2 (1987) 80-94.

SrTo1_87 T. K. Srikanth, Sam Toueg: Optimal Clock Synchronization; Journal of the ACM 34/3 (1987) 626-645.

Tane_88 Andrew S. Tanenbaum: Computer Networks; 2nd ed., Prentice-Hall, Englewood Cliffs 1988.

Toue_84 Sam Toueg: Randomized Byzantine Agreements; 3rd Symposium on Principles of Distributed Systems (PODC) 1984, ACM, New York 1984, 163-178.

WaPf_89 Michael Waidner, Birgit Pfitzmann: Unconditional Sender and Recipient Untraceability in spite of Active Attacks – Some Remarks; Fakultät für Informatik, Universität Karlsruhe, Interner Bericht 5/89, März 1989.

WeCa_81 Mark N. Wegman, J. Lawrence Carter: New Hash Functions and Their Use in Authentication and Set Equality; Journal of Computer and System Sciences 22 (1981) 265-279.

Yao_82 Andrew C. Yao: Protocols for Secure Computations; 23rd Symposium on Foundations of Computer Science (FOCS) 1982, IEEE Computer Society, 1982, 160-164.

Yao_86 Andrew C. Yao: How to Generate and Exchange Secrets; 27th Symposium on Foundations of Computer Science (FOCS) 1986, IEEE Computer Society, 1986, 162-167.

Lebenslauf

Name:	Michael Waidner
Geburtsdatum:	20. Dezember 1961
Geburtsort:	Mühlacker
Staatsangehörigkeit:	Deutsch
Schulbesuch:	1968-1972 Mörike-Grundschule und Haidenwäldle-Grundschule Mühlacker 1972-1981 Theodor-Heuss-Gymnasium Mühlacker 27. Mai 1981 Abitur
Studium:	1981-1986 Informatik mit Ergänzungsfach Mathematik an der Universität Karlsruhe 19. Juli 1983 Diplom-Vorprüfung in Informatik 28. August 1986 Diplom-Hauptprüfung in Informatik
Berufstätigkeit:	Seit 1. September 1986 wissenschaftlicher Mitarbeiter am Institut für Rechnerentwurf und Fehlertoleranz (früher: Institut für Informatik IV) der Universität Karlsruhe, Lehrstuhl Prof. Dr.-Ing. W. Görke